

Package ‘Nestimate’

October 7, 2026

Title Dynamic, Probabilistic, and Higher-Order Network Analysis

Version 0.9.24

Description Estimate, compare, and analyze dynamic and psychological networks using a unified interface. Provides transition network analysis estimation (transition, frequency, co-occurrence, attention-weighted) Saqr et al. (2025) <[doi:10.1145/3706468.3706513](https://doi.org/10.1145/3706468.3706513)>, psychological network methods (correlation, partial correlation, 'graphical lasso', 'Ising') Saqr, Beck, and Lopez-Pernas (2024) <[doi:10.1007/978-3-031-54464-4_19](https://doi.org/10.1007/978-3-031-54464-4_19)>, and higher-order network methods including higher-order networks, higher-order network embedding, hyper-path anomaly, and multi-order generative model. Supports bootstrap inference, permutation testing, split-half reliability, centrality stability analysis, mixed Markov models, multi-cluster multi-layer networks and clustering.

License MIT + file LICENSE

URL <https://github.com/mohsaqr/Nestimate>,
<https://pak.dynasite.org/Nestimate/>

BugReports <https://github.com/mohsaqr/Nestimate/issues>

Language en-US

Encoding UTF-8

RoxygenNote 7.3.3

Imports ggplot2, data.table, cluster, scales, brglm2, nnet,
idiographic (>= 0.3.4), psychnets (>= 0.5.2)

Suggests testthat (>= 3.0.0), igraph, glmnet, lavaan, stringdist,
gridExtra, lme4, corpcor, Matrix, cograph (>= 2.4.4),
ggfitttext, knitr, rmarkdown, tna

Config/testthat/edition 3

Depends R (>= 4.1.0)

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Author Mohammed Saqr [aut, cre, cph],
 Sonsoles López-Pernas [aut],
 Kamila Misiejuk [aut]

Maintainer Mohammed Saqr <saqr@saqr.me>

Repository CRAN

Date/Publication 2026-10-07 16:40:02 UTC

Contents

action_to_onehot	5
actor_endpoints	6
association_rules	7
as_htna	9
as_netdifference	11
as_netobject	12
as_networks	13
as_tna	14
bayes_compare	16
betti_numbers	19
bipartite_groups	20
bootstrap_network	21
boot_glasso	25
bottleneck_distance	29
build_atna	30
build_clusters	31
build_cna	35
build_cor	36
build_ftna	37
build_gimme	38
build_glasso	40
build_hon	41
build_honem	44
build_hypa	46
build_hypergraph	49
build_ising	51
build_mcml	52
build_mcml_pc	56
build_mlvar	62
build_mmm	65
build_mogen	69
build_network	71
build_pcor	77
build_simplicial	78
build_tna	80
casedrop_reliability	81

centrality_stability	85
certainty	88
chain_structure	90
chatgpt_srl	94
clique_expansion	95
cluster_choice	96
cluster_diagnostics	99
cluster_mmm	101
cluster_network	103
cluster_summary	104
coefs	108
compare_mmm	109
compare_model	111
compare_networks	114
comparison_tables	119
composites	120
convert_sequence_format	121
cooccurrence	123
distribution_plot	125
effects_table	128
entropy_bayes	129
entropy_network	131
entropy_trajectory	134
estimate_network	136
euler_characteristic	138
evaluate_links	138
extract_edges	139
extract_initial_probs	140
extract_pathways	141
extract_transition_matrix	143
frequencies	144
get_estimator	146
group_regulation_long	146
hypergraph_centrality	147
hypergraph_cluster	149
hypergraph_laplacian	151
hypergraph_measures	152
hypergraph_transduction	154
item_loadings	157
learning_activities	158
list_estimators	159
loading_stability	159
long-data	161
long_to_wide	162
macro_network	163
magnitude_difference	164
markov_order_test	166
markov_stability	169

mark_first_state	171
mark_terminal_state	172
mogen_transitions	173
mosaic_analysis	174
mosaic_plot	176
nct	180
network_reliability	182
net_aggregate_weights	184
net_centrality	185
net_deprune	188
net_edge_betweenness	189
net_prune	190
net_pruning_details	193
net_reprune	194
outcome_model	195
passage_time	197
pathways	199
path_counts	201
path_dependence	202
permutation	205
permutation_diagnostics	210
persistence_landscape	213
persistent_homology	214
plot_mosaic	216
plot_state_frequencies	217
predictability	222
predict_links	224
prepare	226
prepare_for_tna	228
prepare_onehot	230
q_analysis	231
register_estimator	233
remove_estimator	234
rename_models	234
sequence_compare	235
sequence_plot	238
session_ids	244
set_state_colors	245
simplicial_degree	247
simplicial_features	247
srl_strategies	248
state_colors	249
state_distribution	250
state_frequencies	251
subtract_networks	252
trajectories	253
transition_entropy	253
validate_netobject	256

action_to_onehot

5

verify_simplicial

257

vertex_bootstrap

258

vertex_compare

260

wide_to_long

263

wtna

264

Index

267

action_to_onehot	<i>Convert Action Column to One-Hot Encoding</i>
------------------	--

Description

Convert a categorical Action column to one-hot (binary indicator) columns.

Usage

```
action_to_onehot(  
  data,  
  action_col = "Action",  
  states = NULL,  
  drop_action = TRUE,  
  sort_states = FALSE,  
  prefix = ""  
)
```

Arguments

data	Data frame containing an action column.
action_col	Character. Name of the action column. Default: "Action".
states	Character vector or NULL. States to include as columns. If NULL, uses all unique values. Default: NULL.
drop_action	Logical. Remove the original action column. Default: TRUE.
sort_states	Logical. Sort state columns alphabetically. Default: FALSE.
prefix	Character. Prefix for state column names. Default: "".

Value

The input data frame with one 0/1 integer column appended per state (named paste0(prefix, state)). All other columns are kept; the original action column is removed unless drop_action = FALSE.

Examples

```
long_data <- data.frame(
  Actor = rep(1:3, each = 4),
  Time = rep(1:4, 3),
  Action = sample(c("A", "B", "C"), 12, replace = TRUE)
)
onehot_data <- action_to_onehot(long_data)
head(onehot_data)
```

actor_endpoints	<i>Tidy per-actor endpoint summary of a wide-format sequence dataset</i>
-----------------	--

Description

For each actor (row), reports the first and last observed states, the time indices at which they appear, the number of observed steps, and a dropped_out flag that is TRUE when the actor has a *terminal-NA* pattern (after the final observed step, every remaining cell is NA).

Usage

```
actor_endpoints(data, cols = NULL)
```

Arguments

data	A wide-format matrix or data.frame where rows are actors and columns are time steps. Cells are state labels; NA represents missing observations. If data is a data.frame, non-character/-factor columns (e.g. an id column) are dropped via the cols argument.
cols	Optional character vector of state-column names. If NULL (default) every column is treated as a state column.

Value

A tidy data.frame with one row per actor and columns:

actor	Row number (or row name if present).
first_state	First non-NA state.
last_state	Last non-NA state.
first_step	Column index of the first observed state.
last_step	Column index of the last observed state.
n_observed	Number of non-NA cells.
dropped_out	TRUE iff every cell after last_step is NA and last_step < ncol(data).

See Also

[mark_terminal_state\(\)](#), [chain_structure\(\)](#)

Examples

```
actor_endpoints(trajectories) |> head()
```

association_rules	<i>Discover Association Rules from Sequential or Transaction Data</i>
-------------------	---

Description

Discovers association rules using the Apriori algorithm with proper candidate pruning. Accepts `netobject` (extracts sequences as transactions), data frames, lists, or binary matrices.

Support counting is vectorized via `crossprod()` for 2-itemsets and logical matrix indexing for k-itemsets.

Usage

```
association_rules(
  x,
  min_support = 0.1,
  min_confidence = 0.5,
  min_lift = 1,
  max_length = 5L
)

## S3 method for class 'net_association_rules'
print(x, ...)

## S3 method for class 'net_association_rules'
summary(object, ...)

## S3 method for class 'net_association_rules'
plot(x, ...)
```

Arguments

x	Input data. Accepts: netobject Uses <code>\$data</code> sequences - each sequence becomes a transaction of its unique states. list Each element is a character vector of items (one transaction). data.frame Wide format: each row is a transaction, character columns are item occurrences. Or a binary matrix (0/1). matrix Binary transaction matrix (rows = transactions, columns = items). For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_association_rules</code> .
min_support	Numeric. Minimum support threshold. Default: 0.1.
min_confidence	Numeric. Minimum confidence threshold. Default: 0.5.

<code>min_lift</code>	Numeric. Minimum lift threshold. Default: 1.0.
<code>max_length</code>	Integer. Maximum itemset size. Default: 5.
<code>...</code>	In <code>plot.net_association_rules()</code> : Additional arguments passed to <code>ggplot2</code> functions. In <code>print.net_association_rules()</code> and <code>summary.net_association_rules()</code> : Additional arguments (ignored).
<code>object</code>	For the <code>summary()</code> method: an object of class <code>net_association_rules</code> .

Details

Algorithm:

Uses level-wise Apriori (Agrawal & Srikant, 1994) with the full pruning step: after the join step generates k-candidates, all (k-1)-subsets are verified as frequent before support counting. This is critical for efficiency at $k \geq 4$.

Metrics:

support $P(A \text{ and } B)$. Fraction of transactions containing both antecedent and consequent.

confidence $P(B | A)$. Fraction of antecedent transactions that also contain the consequent.

lift $P(A \text{ and } B) / (P(A) * P(B))$. Values > 1 indicate positive association; < 1 indicate negative association.

conviction $(1 - P(B)) / (1 - \text{confidence})$. Measures departure from independence. Higher = stronger implication.

Value

An object of class "net_association_rules" containing:

rules Tidy data frame, one row per rule, ordered by descending lift then confidence, with columns antecedent and consequent (the itemsets as comma-separated character strings), support, confidence, lift, conviction, count and n_transactions.

frequent Tidy data frame, one row per frequent itemset, with columns itemset, size, support and count.

frequent_itemsets List of frequent itemsets per level k, each entry a list of items / count / support.

items Character vector of the frequent 1-itemsets the mining ran on (all items when no item clears min_support).

n_transactions Integer.

n_rules Integer.

params List of min_support, min_confidence, min_lift, max_length.

In `print.net_association_rules()`: The input object, invisibly.

In `summary.net_association_rules()`: The tidy rules data frame: one row per rule, with columns antecedent, consequent, support, confidence, lift, conviction, count and n_transactions.

In `plot.net_association_rules()`: The drawn ggplot object, invisibly (the plot is also printed). NULL, invisibly, when no rule was found.

Methods

- `plot.net_association_rules()`: Scatter plot of association rules: support vs confidence, with point size proportional to lift.

References

- Agrawal, R. & Srikant, R. (1994). Fast algorithms for mining association rules. In *Proc. 20th VLDB Conference*, 487–499.
- Brin, S., Motwani, R., Ullman, J. D. & Tsur, S. (1997). Dynamic itemset counting and implication rules for market basket data. In *Proc. ACM SIGMOD*, 255–264. (lift and conviction)

See Also

[build_network](#), [predict_links](#)

Examples

```
# From a list of transactions
trans <- list(
  c("plan", "discuss", "execute"),
  c("plan", "research", "analyze"),
  c("discuss", "execute", "reflect"),
  c("plan", "discuss", "execute", "reflect"),
  c("research", "analyze", "reflect")
)
rules <- association_rules(trans, min_support = 0.3, min_confidence = 0.5)
print(rules)

# From a netobject (sequences as transactions)
seqs <- data.frame(
  V1 = sample(LETTERS[1:5], 50, TRUE),
  V2 = sample(LETTERS[1:5], 50, TRUE),
  V3 = sample(LETTERS[1:5], 50, TRUE)
)
net <- build_network(seqs, method = "relative")
rules <- association_rules(net, min_support = 0.1)
```

as_htna

Build a grouped node-level network (htna) from data and a clustering

Description

Builds the full node-level network from the original data and attaches a cluster grouping, producing a single htna network in which every actor is a node and cluster membership labels the actors. This is the node-level counterpart of [build_mcml](#): where `build_mcml` collapses the network to a cluster-level (macro) summary, `as_htna` keeps every node and every transition - including the between-cluster transitions an `mcml` only retains in aggregate.

Usage

```
as_htna(x, clusters = NULL, method = "relative", ...)

## S3 method for class 'mcml'
as_htna(x, clusters = NULL, method = "relative", data = NULL, ...)

## S3 method for class 'net_mmm'
as_htna(x, clusters = NULL, method = "relative", ...)

## Default S3 method:
as_htna(x, clusters = NULL, method = "relative", ...)
```

Arguments

<code>x</code>	Data accepted by build_network (sequence data frame, edgelist, transition matrix, netobject, or tna); or an mcml object, which provides the node-cluster membership and, when it was built from wide sequence data, the retained source as well (see data); or a fitted net_mmm object, which is materialized into one HTNA per sequence cluster using its preserved actor partition.
<code>clusters</code>	Cluster assignment: a named list of node-name vectors, a per-node membership vector, or a two-column data frame. When NULL and x carries node groups (or is an mcml), those are used.
<code>method</code>	Estimator passed to build_network . Default "relative" (row-normalized transitions).
<code>...</code>	Further arguments forwarded to build_network (e.g. actor, action, time for long-format data).
<code>data</code>	For the mcml method, the original data the mcml was built from (sequence/edgelist/etc.). Optional when the mcml was built from wide sequence data (long-format input counts, since build_mcml() widens it first): that source is stashed on the object, so as_htna(mcml) works on its own. Required for an mcml built from a matrix, an aggregate, or an edge list, none of which retain a usable node-level source.

Details

Why this rebuilds from data. An mcml stores cluster-level data (the macro sequences are recoded to cluster labels, and the per-cluster data is filtered to within-cluster nodes), so it does not retain a faithful node-level transition network. The only faithful source of node-level between-cluster transitions is the original data. [as_htna\(\)](#) therefore rebuilds from data via [build_network](#); an mcml supplies the cluster membership and either its retained source or explicitly supplied original data supplies the transitions.

The result is a genuine netobject, so it supports inference ([bootstrap_network](#), centrality, [permutation](#)) and plots directly as a grouped network with cograph: `cograph::plot_htna(as_htna(data, clusters))`.

Value

For data and mcml inputs, a single htna (also a netobject and cograph_network) over all nodes. Cluster labels are stored as a factor in `$nodes$groups` and as character values in `$node_groups$group`;

\$factor_levels records their order and is also attached to \$node_groups for lossless partition round trips. For compatibility, the result also retains \$nodes\$cluster and the membership in the "cluster_members" attribute. A fitted net_mmm returns an htma_group, one materialized HTNA network per sequence cluster, while preserving the MMM diagnostics.

See Also

[build_mcml](#), [build_network](#); plot with `cograph::plot_htna()`.

Examples

```
seqs <- data.frame(
  t1 = c("A", "C", "E", "B"), t2 = c("B", "D", "F", "A"),
  t3 = c("C", "A", "E", "D"), stringsAsFactors = FALSE
)
clusters <- list(C1 = c("A", "B"), C2 = c("C", "D"), C3 = c("E", "F"))
net <- as_htna(seqs, clusters)
net
## Not run:
cograph::plot_htna(net)

## End(Not run)
```

as_netdifference	<i>Coerce an inferential comparison to a network difference</i>
------------------	---

Description

Coerce an inferential comparison to a network difference

Usage

```
as_netdifference(x, ...)

## S3 method for class 'net_bayes'
as_netdifference(x, significant_only = TRUE, ...)

## S3 method for class 'netdifference'
as_netdifference(x, ...)

## Default S3 method:
as_netdifference(x, ...)
```

Arguments

x	An object with network-difference fields.
...	Additional arguments passed to methods.

significant_only

Logical. For inferential objects, keep only supported differences in the plotted weight matrix while retaining the full difference and interval matrices. Default TRUE.

Value

A netdifference object suitable for `cograph::splot()`.

Examples

```
s1 <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
s2 <- data.frame(V1 = c("A", "C", "B"), V2 = c("C", "B", "A"))
b <- bayes_compare(build_network(s1, method = "relative"),
                   build_network(s2, method = "relative"),
                   draws = 500, seed = 1)
as_netdifference(b, significant_only = FALSE)
```

as_netobject

Coerce a network object to a Nestimate netobject

Description

Promotes a **psychnets** result (class `c("psychnet", "cograph_network")`) or any bare `cograph_network` to the dual-class `c("netobject", "cograph_network")` used throughout Nestimate, so it dispatches to the package's verbs (`centrality()`, `plot()`, `bootstrap`, `reliability`, ...). A `netobject` is returned unchanged.

Usage

```
as_netobject(x)

## S3 method for class 'netobject'
as_netobject(x)

## S3 method for class 'psychnet'
as_netobject(x)

## S3 method for class 'cograph_network'
as_netobject(x)

## Default S3 method:
as_netobject(x)
```

Arguments

x A psychnet object, a cograph_network, or a netobject.

Details

The psychnet method re-derives the integer-indexed edge table that Nestimate expects (psychnet stores character-labelled edges), preserves the estimator name in `$method`, and parks every psychnet-specific field - including the graphical-lasso `$kkt` optimality certificate - under `$meta$psychnet` so nothing is lost in translation.

Value

A `c("netobject", "cograph_network")` object.

See Also

[validate_netobject](#)

Examples

```
net <- build_cor(data.frame(a = rnorm(50), b = rnorm(50), c = rnorm(50)))
identical(as_netobject(net), net) # netobjects pass through unchanged
```

as_networks

Promote a psychometric MCML result to a network group

Description

`as_networks()` is the psychometric-network counterpart of [as_tna](#). It promotes the cluster-level (macro) and within-cluster networks produced by [build_mcml_pc](#) into a single `netobject_group`, so the result flows into the same downstream verbs as any other group of networks (`print()`, `summary()`, `plot()`, [net centrality](#)).

Usage

```
as_networks(x)

## S3 method for class 'mcml_pc'
as_networks(x)

## Default S3 method:
as_networks(x)
```

Arguments

`x` An object to convert. The `mcml_pc` method (from [build_mcml_pc](#)) is the primary path.

Details

Where `as_tna()` promotes *transition* networks (directed, row-normalised, with initial probabilities) and re-wraps raw matrices, `as_networks()` promotes *psychometric* networks (undirected; correlation / partial-correlation / glasso). The macro and within-cluster components of an `mcml_pc` object are already full netobjects carrying their estimator, directedness and data, so this function assembles them into a group rather than re-wrapping matrices.

Value

A `netobject_group`: a named list whose first element is macro (the cluster-level network), followed by one netobject per non-singleton cluster.

The `mcml_pc` method returns a `netobject_group`; singleton clusters (no within-network) are dropped with a warning().

The default method returns the input unchanged if it is already a `netobject_group`, otherwise it errors.

See Also

[build_mcml_pc](#) to create the input, [as_tna](#) for the transition-network counterpart.

Examples

```
set.seed(1)
f <- stats::rnorm(200)
g <- stats::rnorm(200)
df <- data.frame(a1 = f + stats::rnorm(200), a2 = f + stats::rnorm(200),
                 a3 = f + stats::rnorm(200), b1 = g + stats::rnorm(200),
                 b2 = g + stats::rnorm(200), b3 = g + stats::rnorm(200))
clusters <- list(A = c("a1", "a2", "a3"), B = c("b1", "b2", "b3"))
fit <- build_mcml_pc(df, clusters, aggregation = "composite", method = "cor")
nets <- as_networks(fit)
nets
summary(nets)
```

as_tna

Promote the Layers of an mcml to Networks

Description

Converts an `mcml` object into a `netobject_group`: one netobject for the cluster-level (macro) layer, and one per cluster for the within-cluster layers. The stored weights are carried over as they are – nothing is re-normalised here, so the aggregation chosen when the `mcml` was built is what the networks hold.

Usage

```
as_tna(x, ...)

## S3 method for class 'mcml'
as_tna(x, expand = NULL, ...)

## Default S3 method:
as_tna(x, ...)
```

Arguments

x	An mcml object created by cluster_summary or build_mcml .
...	Passed to methods.
expand	For the mcml method, names of clusters whose member states replace the collapsed cluster node in the macro layer (see macro_network). NULL (default) keeps the macro fully collapsed. The per-cluster layers are unaffected.

Details

This is the step that lets an MCML result flow into the verbs that take a group of networks (printing, network-metric summaries, rendering with cograph).

Workflow:

```
# Full MCML workflow
net <- build_network(data, method = "relative")
cs  <- cluster_summary(net, clusters = group_assignments)
nets <- as_tna(cs)
```

```
# Every layer is an ordinary netobject
print(nets)      # one line per layer
summary(nets)    # network metrics per layer
```

Zero-out-degree (sink) nodes:

Every cluster is returned, regardless of its row sums. A node with zero outgoing weight is a legitimate sink (a terminal state); its row in the wrapped network is left all-zero. This holds for both `net_method = "relative"` and `"frequency"` – the stored weights are never re-normalised, so a sink row needs no special handling. Inspect `rowSums(x$clusters[[cl]]$weights)` to find sink nodes.

Value

A `netobject_group`: a named list whose first element is `macro` (the $k \times k$ cluster-level network) followed by one element per cluster, each a `netobject/cograph_network` carrying `$weights`, `$inits`, `$nodes`, `$edges` and the recorded `$method` (`"relative"` for an mcml whose weights are already row-normalised, `"frequency"` otherwise).

The mcml method returns that `netobject_group`, each layer keeping the data the corresponding mcml layer carried. With `expand`, its `macro` element is the mixed-resolution network of [macro_network](#) rather than the fully collapsed one.

The default method returns the input unchanged when it already inherits from `tna`, and otherwise raises an error.

See Also

`cluster_summary` and `build_mcml` to create the input object, `macro_network` for a macro layer with one cluster expanded, `as_networks` for the psychometric-network counterpart

Examples

```
set.seed(1)
mat <- matrix(runif(36), 6, 6)
rownames(mat) <- colnames(mat) <- LETTERS[1:6]
clusters <- list(G1 = c("A", "B"), G2 = c("C", "D"), G3 = c("E", "F"))
cs <- cluster_summary(mat, clusters)
nets <- as_tna(cs)
nets
summary(nets)
```

bayes_compare	<i>Bayesian Dirichlet-Multinomial comparison of two transition networks</i>
---------------	---

Description

Compares two transition networks estimated by `build_network` (method "relative" or "frequency") using a Bayesian Dirichlet-Multinomial model. The outgoing transitions from each source state are modelled as a Multinomial draw with a Dirichlet prior on the transition probabilities. With a Jeffreys prior the posterior for the transitions out of state i is $\text{Dirichlet}(c_i + \alpha)$, where c_i are the observed outgoing counts. Each edge probability is then marginally Beta-distributed, so the posterior mean difference between the two networks is available in closed form and a credible interval is obtained by Monte Carlo.

This is a complement to `permutation`: the permutation test answers "is this difference more extreme than chance?"; the Bayesian comparison answers "what is the plausible range of the true difference, and how precisely is it estimated given the counts?". An edge with few outgoing transitions from its source state yields a wide credible interval even when its row-normalised probability looks decisive.

`bayes_compare()` also accepts two `net_edge_betweenness` objects (source method "relative" only). Edge betweenness is a nonlinear function of the whole transition matrix, so instead of Beta marginals the full transition matrix is drawn from each group's row-wise Dirichlet posterior and edge betweenness is recomputed on every draw - the Bayesian analogue of `permutation()`'s edge-betweenness dispatch. The result summarises the posterior of $\text{EB}(x) - \text{EB}(y)$: `diff` is the posterior mean difference, `prob_x/prob_y` hold the posterior mean betweenness matrices, and `observed_diff` the plug-in difference of the two input networks. Both inputs must use the same `invert` setting.

Usage

```

bayes_compare(
  x,
  y = NULL,
  prior = 0.5,
  draws = 10000L,
  ci = 0.95,
  mean_threshold = 0.01,
  bound_threshold = 0.001,
  seed = NULL
)

## S3 method for class 'net_bayes'
print(x, ...)

## S3 method for class 'net_bayes'
summary(object, ...)

## S3 method for class 'net_bayes'
plot(x, significant_only = TRUE, title = NULL, ...)

## S3 method for class 'net_bayes_group'
print(x, ...)

## S3 method for class 'net_bayes_group'
summary(object, ...)

```

Arguments

<code>x</code>	A netobject (from build_network), a netobject_group, an mcmc object, or a net_edge_betweenness object. Must use a transition method ("relative" / "frequency" and their aliases). For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_bayes</code> or <code>net_bayes_group</code> .
<code>y</code>	A second object of the same kind as <code>x</code> , or <code>NULL</code> . When <code>x</code> is a <code>netobject_group</code> and <code>y</code> is <code>NULL</code> , all pairwise comparisons among the groups are returned.
<code>prior</code>	Numeric. Dirichlet prior concentration added to every cell (default 0.5, the Jeffreys prior). Use 1 for a uniform (Laplace) prior.
<code>draws</code>	Integer. Number of Monte Carlo posterior draws used for the credible intervals (default 10000).
<code>ci</code>	Numeric in (0, 1). Credible interval mass (default 0.95).
<code>mean_threshold</code>	Numeric. An edge is flagged significant only if the absolute posterior mean difference exceeds this (default 0.01).
<code>bound_threshold</code>	Numeric. An edge is flagged significant only if the credible-interval bound nearest zero exceeds this in absolute value (default 0.001). Guards against differences that are detectable but negligibly small.

<code>seed</code>	Integer or NULL. RNG seed for reproducible credible intervals.
<code>...</code>	In <code>plot.net_bayes()</code> : Additional arguments passed to <code>cograph::splot()</code> when <code>cograph</code> is available. In <code>print.net_bayes()</code> , <code>print.net_bayes_group()</code> , <code>summary.net_bayes()</code> and <code>summary.net_bayes_group()</code> : Additional arguments (ignored).
<code>object</code>	For the <code>summary()</code> method: an object of class <code>net_bayes</code> or <code>net_bayes_group</code> .
<code>significant_only</code>	Logical. Show only credibly-different edges (default TRUE).
<code>title</code>	Optional plot title.

Value

An object of class `c("net_bayes", "netdifference", "net_permutation")`. It carries the same fields as a [permutation](#) result, so it is a drop-in wherever a `net_permutation` is consumed, and also carries a `netdifference` difference matrix for `cograph` difference plotting, plus Bayesian extras:

x, y The two input `netobjects`.

diff Posterior mean difference matrix (`prob_x - prob_y`); the analogue of the permutation observed difference.

difference_matrix Alias of `diff` for `cograph` `netdifference` helpers.

diff_sig Difference where `sig`, else 0.

p_values The two-sided Bayesian p-equivalent, in the field a `net_permutation` consumer reads as p-values (see `p_bayes`).

effect_size Posterior mean difference over its posterior SD.

ci_lower, ci_upper Credible-interval bound matrices.

p_difference Probability of the difference: the share of posterior mass on the dominant side of zero, in $[0.5, 1]$ ($P(\text{High} > \text{Low})$ for a positive difference).

p_bayes Alias of `p_values`: the two-sided Bayesian *p-equivalent* $2(1 - \text{p_difference})$. It summarises posterior mass, not a frequentist tail probability, so it is not a p-value and should not be reported as one.

prob_x, prob_y Posterior mean transition-probability matrices.

sig Logical significance matrix (CI excludes zero, mean and nearest bound exceed their thresholds).

summary Long-format data frame whose columns are a superset of `summary.net_permutation` (from, to, weight_x, weight_y, diff, effect_size, p_value, sig) plus `count_x`, `count_y`, `ci_lower`, `ci_upper`, `ci_width`, `p_difference`.

method, iter, alpha, paired, adjust Permutation-compatible settings (`iter = draws`, `alpha = 1 - ci`, `paired = FALSE`, `adjust = "none"`).

prior, draws, ci, mean_threshold, bound_threshold Bayesian settings.

In `print.net_bayes()` and `print.net_bayes_group()`: The input object, invisibly.

In `summary.net_bayes()`: A data frame with edge-level posterior differences and intervals.

In `plot.net_bayes()`: Invisibly, the `cograph` network returned by `cograph::splot()` when `cograph` is available; otherwise a fallback `ggplot` object.

In `summary.net_bayes_group()`: A combined data frame with a comparison column.

Methods

- `plot.net_bayes()`: Draws a differential transition network as a directed chord diagram. Edge colour encodes the signed posterior mean difference (x stronger vs y stronger) and edge width its magnitude.

References

- Johnston, L. & Jendoubi, T. (2026). How Delivery Mode Reshapes Resource Engagement: A Bayesian Differential Network Analysis. TNA Workshop 2026.
- Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian Data Analysis* (3rd ed.). CRC Press.
- Jeffreys, H. (1946). An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London A*, 186(1007), 453-461.

See Also

[permutation](#) for the frequentist complement; [certainty](#) for single-network posterior edge intervals; [subtract_networks](#) and [as_netdifference](#) for the difference verbs; [build_network](#)

Examples

```
s1 <- data.frame(V1 = c("A","B","C"), V2 = c("B","C","A"))
s2 <- data.frame(V1 = c("A","C","B"), V2 = c("C","B","A"))
n1 <- build_network(s1, method = "relative")
n2 <- build_network(s2, method = "relative")
bayes_compare(n1, n2, draws = 500, seed = 1)
```

betti_numbers

Betti Numbers

Description

Computes Betti numbers: β_0 (components), β_1 (loops), β_2 (voids), etc.

Usage

```
betti_numbers(sc)
```

Arguments

sc A simplicial_complex object.

Value

Named integer vector `c(b0 = ..., b1 = ..., ...)`.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
betti_numbers(sc)
```

bipartite_groups	<i>Hypergraph from bipartite group / event data</i>
------------------	---

Description

Constructs a [net_hypergraph](#) from long-format event data in which each row records a player participating in a group (a session, team, project, transaction, or any group context). Each unique group becomes one hyperedge spanning the players that appeared in it. Optional weight column produces a weighted incidence matrix.

Usage

```
bipartite_groups(data, player, group, weight = NULL)
```

Arguments

data	Data frame in long format. Must contain player and group columns; optionally a weight column.
player	Character. Name of the column whose values become the hypergraph's nodes (players, participants, actors).
group	Character. Name of the column whose values become the hypergraph's hyperedges (groups, sessions, teams).
weight	Character or NULL. If supplied, the column is summed per (player, group) pair to produce a weighted incidence matrix. Default NULL produces a 0/1 binary incidence matrix.

Details

The bipartite representation preserves the full group structure without projecting to a pairwise network. A group of three players A, B, C produces a single 3-hyperedge containing all three, not three pairwise edges AB, AC, BC. This avoids information loss when group interactions are the primary unit of analysis (Perc et al. 2013).

Unlike [build_hypergraph\(\)](#) (which derives hyperedges from a network's clique structure), `bipartite_groups()` takes group memberships directly. The two functions are complementary:

- `bipartite_groups()` - when group membership is observed (sessions, transactions, co-authorships).
- `build_hypergraph()` - when only pairwise interactions are observed and triadic structure must be inferred from triangles.

Rows with NA in either the player or group column (or, when supplied, the weight column) are dropped silently.

Value

A net_hypergraph object with the same structure produced by `build_hypergraph()` (hyperedges, incidence, nodes, n_nodes, n_hyperedges, size_distribution, params). The params list records source = "bipartite_groups" and the original column names.

Note

(experimental) Validated against a hand-computed `table()` incidence reference only; no independent R package exposes the long-format-to-binary-incidence primitive, because the operation is definitionally `table()`. The code path is a direct one-to-one restatement of its definition.

References

Perc, M., Gomez-Gardenes, J., Szolnoki, A., Floria, L. M., & Moreno, Y. (2013). Evolutionary dynamics of group interactions on structured populations: a review. *Journal of the Royal Society Interface* 10(80), 20120997. doi:10.1098/rsif.2012.0997

See Also

`build_hypergraph()` for the clique-based constructor.

Examples

```
df <- data.frame(
  player = c("Alice", "Bob", "Carol", "Alice", "Bob",
            "Dave", "Carol", "Dave", "Eve"),
  session = c("S1", "S1", "S1", "S2", "S2",
            "S3", "S3", "S3", "S3")
)
hg <- bipartite_groups(df, player = "player", group = "session")
print(hg)
summary(hg)
```

bootstrap_network

*Bootstrap a Network Estimate***Description**

Non-parametric bootstrap for any network estimated by `build_network`. Works with all built-in methods (transition and association) as well as custom registered estimators.

For transition methods ("relative", "frequency", "co_occurrence"), uses a fast pre-computation strategy: per-sequence count matrices are computed once, and each bootstrap iteration only resamples sequences via `colSums` (C-level) plus lightweight post-processing. Data must be in wide format for transition bootstrap; use `convert_sequence_format` to convert long-format data first.

For association methods ("cor", "pcor", "glasso", and custom estimators), the full estimator is called on resampled rows each iteration.

If a transition network contains only one sequence, the function warns that such a network is not recommended for bootstrap or other confirmatory testing.

Usage

```
bootstrap_network(
  x,
  iter = 1000L,
  ci_level = 0.05,
  inference = "stability",
  consistency_range = c(0.75, 1.25),
  edge_threshold = NULL,
  seed = NULL,
  boundary = c("inclusive", "strict"),
  ci_method = c("percentile", "basic"),
  actor = NULL
)

## S3 method for class 'net_bootstrap'
print(x, ...)

## S3 method for class 'net_bootstrap'
summary(object, ...)

## S3 method for class 'net_bootstrap_group'
print(x, ...)

## S3 method for class 'net_bootstrap_group'
summary(object, ...)

## S3 method for class 'wtna_boot_mixed'
print(x, ...)

## S3 method for class 'wtna_boot_mixed'
summary(object, ...)
```

Arguments

<code>x</code>	A netobject from build_network . The data, method, params, scaling, threshold, and level are all extracted from this object. A <code>cograph_network</code> is coerced first; a <code>netobject_group</code> or <code>mcml</code> bootstraps every constituent network, and a <code>wtna_mixed</code> bootstraps both of its components (see Value). For the <code>print()</code> method: an object of class <code>net_bootstrap</code> , <code>net_bootstrap_group</code> or <code>wtna_boot_mixed</code> .
<code>iter</code>	Integer. Number of bootstrap iterations (default: 1000).
<code>ci_level</code>	Numeric. Significance level for CIs and p-values (default: 0.05).
<code>inference</code>	Character. "stability" (default) tests whether bootstrap replicates fall within a multiplicative consistency range around the original weight. "threshold" tests whether replicates exceed a fixed edge threshold.
<code>consistency_range</code>	Numeric vector of length 2. Multiplicative bounds for stability inference (default: <code>c(0.75, 1.25)</code>).

edge_threshold	Numeric or NULL. Fixed threshold for inference = "threshold". If NULL, defaults to the 10th percentile of absolute original edge weights.
seed	Integer or NULL. RNG seed for reproducibility.
boundary	Character. Comparison rule when computing the consistency-range p-value. "inclusive" (default, tna-compatible) counts iterations that meet the bound ($\leq / \geq$); "strict" counts only iterations strictly outside ($< / >$).
ci_method	Character. Method for the edge-weight confidence intervals. "percentile" (default) uses the empirical bootstrap quantiles (Efron). "basic" reflects those quantiles around the observed weight, $(2\hat{\theta} - q_{1-\alpha/2}, 2\hat{\theta} - q_{\alpha/2})$ (Davison & Hinkley 1997, eq. 5.6), which corrects first-order bootstrap bias but can produce bounds outside the natural weight range near boundaries (e.g., below 0 for transition probabilities close to 0).
actor	Character or NULL. Name of the column identifying the actor each sequence belongs to (e.g. "student_id" for sessions nested in students, "Group" for students nested in teams), looked up in the network's \$metadata or wide sequence data. When supplied, whole actors are resampled; see the section <i>Nested data and actor</i> . Default NULL: sequences are resampled individually.
...	In print.net_bootstrap(), print.wtna_boot_mixed(), summary.net_bootstrap() and summary.wtna_boot_mixed(): Additional arguments (ignored). In print.net_bootstrap_group() and summary.net_bootstrap_group(): Ignored.
object	For the summary() method: an object of class net_bootstrap, net_bootstrap_group or wtna_boot_mixed.

Value

An object of class "net_bootstrap" containing:

original The original netobject.

mean Bootstrap mean weight matrix.

sd Bootstrap SD matrix.

p_values P-value matrix.

significant Original weights where $p < ci_level$, else 0.

ci_lower Lower CI bound matrix.

ci_upper Upper CI bound matrix.

cr_lower Consistency range lower bound (stability only).

cr_upper Consistency range upper bound (stability only).

summary Long-format data frame, one row per non-zero original edge (undirected networks keep one row per unordered pair), with columns from, to, weight, mean, sd, p_value, sig, ci_lower, ci_upper, plus cr_lower and cr_upper when inference = "stability".

model Pruned netobject (non-significant edges zeroed).

method, params, iter, ci_level, inference, ci_method Bootstrap config.

consistency_range, edge_threshold Inference parameters.

actor, n_actors The actor column and its number of actors; NULL without actor.

clustering Only with actor. One-row data frame: n_sequences, n_actors, icc, icc_ci_lower, icc_ci_upper, deff_edges.

clustering_edges Only with actor. One row per edge of summary: from, to, icc, sd_actor, sd_sequence, deff.

A `netobject_group` or `mcml` input returns a "net_bootstrap_group" (named list of `net_bootstrap` results); a `wtna_mixed` input returns a "wtna_boot_mixed" with `$transition` and `$cooccurrence` results.

In `print.net_bootstrap()` and `print.wtna_boot_mixed()`: The input object, invisibly.

In `summary.net_bootstrap()`: The `$summary` data frame: one row per non-zero original edge, with columns `from`, `to`, `weight`, `mean`, `sd`, `p_value`, `sig`, `ci_lower`, `ci_upper`, plus `cr_lower` and `cr_upper` when the bootstrap used `inference = "stability"`.

In `print.net_bootstrap_group()`: x invisibly.

In `summary.net_bootstrap_group()`: The per-group summaries stacked into one data frame: the columns of `summary.net_bootstrap` prefixed by a group column naming the network each row came from.

In `summary.wtna_boot_mixed()`: A list with `$transition` and `$cooccurrence` summary data frames.

Nested data and actor

The bootstrap resamples sequences as independent units. When sequences are nested in actors (sessions in students, students in teams), `actor` names the column identifying the actor, and whole actors are resampled with replacement, keeping all their sequences together: the cluster bootstrap that resamples at the top level only (Davison & Hinkley, 1997, section 3.8; Field & Welsh, 2007). The number of sequences per replicate then varies with the actors drawn.

With `actor`, the result also reports the nesting effect. The ICC is the proportion of the total variance that lies between actors (Shrout & Fleiss, 1979); an ICC close to 0 indicates little evidence of a nesting effect. It is computed as in [permutation](#). The design effect is the ratio of the variance under the nested design to the variance had the sequences been sampled independently (Kish, 1965): here, the variance of the edge weights over actor-level replicates divided by their variance over sequence-level replicates drawn in the same run, reported as the median over edges. `actor` is available for transition networks ("relative", "frequency", "co_occurrence").

References

- Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap Methods and Their Application*. Cambridge University Press.
- Field, C. A., & Welsh, A. H. (2007). Bootstrapping clustered data. *Journal of the Royal Statistical Society: Series B*, 69(3), 369-390.
- Kish, L. (1965). *Survey Sampling*. Wiley.
- Shrout, P. E., & Fleiss, J. L. (1979). Intraclass correlations: Uses in assessing rater reliability. *Psychological Bulletin*, 86(2), 420-428.

See Also

[certainty](#) for the closed-form Bayesian counterpart (same result layout, no resampling); [build_network](#), [print.net_bootstrap](#), [summary.net_bootstrap](#)

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C"), V2 = c("B","C","A")),
  method = "relative")
boot <- bootstrap_network(net, iter = 10)

set.seed(1)
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
boot <- bootstrap_network(net, iter = 100)
print(boot)
summary(boot)

# Students nested in teams: resample whole teams
teams <- build_network(group_regulation_long, method = "relative",
  actor = "Actor", action = "Action", time = "Time",
  group = "Achiever")
bootstrap_network(teams, iter = 100, actor = "Group", seed = 1)
```

boot_glasso

*Bootstrap for Regularized Partial Correlation Networks***Description**

Fast, single-call bootstrap for EBICglasso partial correlation networks. Combines nonparametric edge/centrality bootstrap, case-dropping stability analysis, edge/centrality difference tests, predictability CIs, and thresholded network into one function. Designed as a faster alternative to boot-net with richer output.

Usage

```
boot_glasso(
  x,
  iter = 1000L,
  cs_iter = 500L,
  cs_drop = seq(0.1, 0.9, by = 0.1),
  alpha = 0.05,
  gamma = 0.5,
  nlambdas = 100L,
```

```

    centrality = c("strength", "expected_influence", "betweenness", "closeness"),
    centrality_fn = NULL,
    cor_method = "pearson",
    ncores = 1L,
    seed = NULL
)

## S3 method for class 'boot_glasso'
print(x, ...)

## S3 method for class 'boot_glasso'
summary(object, type = "edges", ...)

## S3 method for class 'boot_glasso'
plot(x, type = "edges", measure = NULL, ...)

```

Arguments

<code>x</code>	A data frame, numeric matrix (observations x variables), or a netobject with method = "glasso". For the print() and plot() methods: an object of class boot_glasso.
<code>iter</code>	Integer. Number of nonparametric bootstrap iterations (default: 1000).
<code>cs_iter</code>	Integer. Total number of case-dropping iterations (default: 500). Following bootnet , each iteration draws one drop proportion at random from cs_drop, so the iterations are spread across the proportions rather than repeated cs_iter times at each one.
<code>cs_drop</code>	Numeric vector. Drop proportions for CS-coefficient computation (default: seq(0.1, 0.9, by = 0.1)).
<code>alpha</code>	Numeric. Significance level for CIs (default: 0.05).
<code>gamma</code>	Numeric. EBIC hyperparameter (default: 0.5).
<code>nlambda</code>	Integer. Number of lambda values in the regularization path (default: 100).
<code>centrality</code>	Character vector. Centrality measures to compute. All four built-in measures ("strength", "expected_influence", "betweenness", "closeness") are computed internally with no extra dependencies and are always taken from the built-in path even if centrality_fn is supplied. Names that are not one of these four are valid only when a centrality_fn is supplied; that function is then responsible for returning them. Default: c("strength", "expected_influence", "betweenness", "closeness").
<code>centrality_fn</code>	Optional function. A custom centrality function that takes a weight matrix and returns a named list of centrality vectors. When NULL (default), all four built-in measures are computed internally: "strength"/"expected_influence" via rowSums, and "betweenness"/"closeness" via an internal Floyd-Warshall shortest-path routine. When provided, the function is called as centrality_fn(mat) and is used only for requested measures that are not one of the four built-ins; it should return a named list (e.g., list(my_metric = ...)).
<code>cor_method</code>	Character. Correlation method: "pearson" (default), "spearman", or "kendall".

ncores	Integer. Number of parallel cores for mclapply (default: 1, sequential).
seed	Integer or NULL. RNG seed for reproducibility.
...	In plot.boot_glasso(): Additional arguments passed to plotting functions. For type = "edge_diff" and type = "centrality_diff", accepts order: "sample" (default, sorted by value) or "id" (alphabetical). In print.boot_glasso() and summary.boot_glasso(): Additional arguments (ignored).
object	For the summary() method: an object of class boot_glasso.
type	In summary.boot_glasso(): Character. Summary type: "edges" (default), "centrality", "cs", "predictability", or "all". In plot.boot_glasso(): Character. Plot type: "edges" (default), "stability", "edge_diff", "centrality_diff", or "inclusion".
measure	Character. Centrality measure for type = "centrality_diff" (default: first available measure).

Value

An object of class "boot_glasso" containing:

original_pcor Original partial correlation matrix.

original_precision Original precision matrix.

original_centrality Named list of original centrality vectors.

original_predictability Named numeric vector of node R-squared.

edge_ci Data frame of edge CIs (edge, weight, ci_lower, ci_upper, inclusion).

edge_inclusion Named numeric vector of edge inclusion probabilities.

thresholded_pcor Partial correlation matrix with non-significant edges zeroed.

centrality_ci Named list of data frames (node, value, ci_lower, ci_upper) per centrality measure.

cs_coefficient Named numeric vector of CS-coefficients per centrality measure.

cs_data Data frame of case-dropping results, one row per drop proportion by measure, with columns drop_prop, measure, mean_cor, prop_above (fraction of that proportion's iterations correlating above 0.7) and n_samples (iterations that landed on that proportion).

edge_diff_p Symmetric matrix of pairwise edge difference p-values; NULL when the network has more than 500 edges.

centrality_diff_p Named list of symmetric p-value matrices per centrality measure.

predictability_ci Data frame of node predictability CIs (node, r2, ci_lower, ci_upper).

boot_edges iter x n_edges matrix of bootstrap edge weights.

boot_centrality Named list of iter x p bootstrap centrality matrices.

boot_predictability iter x p matrix of bootstrap R-squared.

nodes Character vector of node names.

n Sample size.

p Number of variables.

iter Number of nonparametric iterations.

cs_iter Number of case-dropping iterations.
cs_drop Drop proportions used.
alpha Significance level.
gamma EBIC hyperparameter.
nlambda Lambda path length.
centrality_measures Character vector of centrality measures.
cor_method Correlation method.
lambda_path Lambda sequence used.
lambda_selected Selected lambda for original data.
timing Named numeric vector with timing in seconds.

In `print.boot_glasso()`: The input object, invisibly.

In `summary.boot_glasso()`: For `type = "edges"`, the `edge_ci` data frame (edge, weight, ci_lower, ci_upper, inclusion) ordered by decreasing absolute weight; for `"cs"` the `cs_data` data frame; for `"predictability"` the `predictability_ci` data frame; for `"centrality"` a named list of one data frame per measure (node, value, ci_lower, ci_upper); for `"all"` a named list holding all four.

In `plot.boot_glasso()`: A ggplot object (returned, and so printed when the call is made at the top level).

Methods

- `plot.boot_glasso()`: Plots bootstrap results for GLASSO networks.

References

Epskamp, S., Borsboom, D., & Fried, E. I. (2018). Estimating psychological networks and their accuracy: A tutorial paper. *Behavior Research Methods* 50(1), 195-212. doi:10.3758/s13428-01708621 (source of the CS-coefficient and of the case-dropping, edge-difference and centrality-difference procedures reproduced here.)

See Also

[build_network](#), [bootstrap_network](#)

Examples

```
set.seed(1)
dat <- as.data.frame(matrix(rnorm(60), ncol = 3))
net <- build_network(dat, method = "glasso")
bg <- boot_glasso(net, iter = 10, cs_iter = 5, centrality = "strength")

set.seed(42)
mat <- matrix(rnorm(60), ncol = 4)
colnames(mat) <- LETTERS[1:4]
net <- build_network(as.data.frame(mat), method = "glasso")
# iter = 20 keeps the example fast; a real analysis uses 1000 or more.
```

```
boot <- boot_glasso(net, iter = 20, cs_iter = 10, seed = 42,
  centrality = c("strength", "expected_influence"))
print(boot)
summary(boot, type = "edges")
```

bottleneck_distance *Bottleneck Distance Between Persistence Diagrams*

Description

Computes the bottleneck distance between two persistence diagrams. For finite pairs, the bottleneck distance is

$$W_{\infty}(D_1, D_2) = \inf_{\gamma} \sup_{p \in D_1} \|p - \gamma(p)\|_{\infty},$$

where γ ranges over bijections $D_1 \cup \Delta \rightarrow D_2 \cup \Delta$ and $\Delta = \{(x, x)\}$ is the diagonal. Each point may match a point in the other diagram or its projection onto the diagonal at cost $|d - b|/2$. Computed via binary search on ε plus a Kuhn bipartite-matching feasibility check.

Essential classes (death = Inf in VR mode, or death = 0 in clique mode) are matched one-to-one within each dimension. If the diagrams have different numbers of essential classes in some dimension, the bottleneck distance for that dimension is Inf.

Usage

```
bottleneck_distance(d1, d2, dimension = NULL, tol = .Machine$double.eps^0.5)
```

Arguments

d1, d2	persistent_homology objects, or data.frames with columns dimension, birth, death.
dimension	Integer vector of dimensions to compare. NULL (default) compares all dimensions appearing in either diagram and returns a named numeric vector.
tol	Numerical tolerance for binary search (default <code>.Machine\$double.eps ^ 0.5</code>).

Value

Named numeric vector. Names are "dim_<k>". Inf indicates a structural mismatch (different essential counts in that dimension); a self-distance is always 0.

References

Edelsbrunner, H. & Harer, J. (2010). *Computational Topology: An Introduction*. AMS. Section VIII.

Examples

```
mat1 <- matrix(c(0, .6, .5, .6, 0, .4, .5, .4, 0), 3, 3)
rownames(mat1) <- colnames(mat1) <- c("A", "B", "C")
ph1 <- persistent_homology(mat1, n_steps = 5)
bottleneck_distance(ph1, ph1) # self-distance is 0
```

build_atna

Build an Attention-Weighted Transition Network (ATNA)

Description

Convenience wrapper for `build_network(method = "attention")`. Computes decay-weighted transitions from sequence data.

Usage

```
build_atna(data, start = FALSE, end = FALSE, ...)
```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance). A fitted <code>net_clustering</code> or <code>net_mmm</code> object is also accepted: the per-cluster networks are (re)built and a <code>netobject_group</code> is returned.
start	Boundary marker prepended to every sequence as an explicit start state (a pure source: no incoming edges, every sequence's first transition is start -> first_observed). FALSE (default) adds nothing; TRUE uses the label "Start"; a single string uses that string as the label. Only valid for the transition methods (relative, frequency, co_occurrence, attention, ngram, gap, reverse); errors otherwise (wtna included).
end	Boundary marker placed in the single cell after each sequence's last observed (non-NA) state, as an explicit terminal state (a pure sink: no outgoing edges, no self-loop – distinct from <code>mark_terminal_state</code> , which fills all trailing NAs into an absorbing state). FALSE (default) adds nothing; TRUE uses the label "End"; a single string uses that string as the label. Same method restriction as start.
...	Additional arguments passed to <code>build_network</code> .

Value

A `netobject` (see `build_network`).

See Also

[build_network](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
net <- build_atna(seqs)
```

 build_clusters

Cluster Sequences by Dissimilarity

Description

Clusters wide-format sequences using pairwise string dissimilarity and either PAM (Partitioning Around Medoids) or hierarchical clustering. Supports 9 distance metrics including temporal weighting for Hamming distance. When the **stringdist** package is available, uses C-level distance computation for 100-1000x speedup on edit distances.

Usage

```
build_clusters(
  data,
  k,
  dissimilarity = "hamming",
  method = "pam",
  na_syms = c("*", "%"),
  weighted = FALSE,
  lambda = 1,
  seed = NULL,
  q = 2L,
  p = 0.1,
  covariates = NULL,
  estimator = c("auto", "firth", "multinom", "chisq"),
  ...
)

## S3 method for class 'net_clustering'
print(x, digits = 3L, ...)

## S3 method for class 'net_clustering'
summary(object, ...)

## S3 method for class 'net_clustering'
plot(
  x,
  type = c("silhouette", "mds", "heatmap", "predictors"),
  combined = TRUE,
  ...
)

## S3 method for class 'tidy_covariates'
print(x, ...)
```

Arguments

data	Input data. Accepts multiple formats: data.frame / matrix Wide-format sequences (rows = sequences, columns = time points, values = state names). netobject A network object from build_network. Extracts the stored sequence data. Only valid for sequence-based methods (relative, frequency, co_occurrence, attention). tna A tna model from the tna package. Decodes the integer-encoded sequence data using stored labels. cograph_network A cograph network object. Extracts the stored sequence data.
k	Integer. Number of clusters (must be between 2 and <code>nrow(data) - 1</code>).
dissimilarity	Character. Distance metric. One of "hamming", "osa" (optimal string alignment), "lv" (Levenshtein), "dl" (Damerau-Levenshtein), "lcs" (longest common subsequence), "qgram", "cosine", "jaccard", "jw" (Jaro-Winkler). Default: "hamming".
method	Character. Clustering method. "pam" for Partitioning Around Medoids, or a hierarchical method: "ward.D2", "ward.D", "complete", "average", "single", "mcquitty", "median", "centroid". Default: "pam".
na_syms	Character vector. Symbols treated as missing values. Default: <code>c("*", "%")</code>. Missing-value distance rule: after symbols are converted to NA, missing values are encoded as a single comparable sentinel state – <i>not</i> pairwise-deleted. Two missing values in the same position match (distance contribution 0); a missing value paired with any observed state mismatches (distance contribution 1 for Hamming, etc.). This is the conventional behaviour for aligned sequence matrices because pairwise deletion would change the effective length of every pair and break the metric. If you want pairwise deletion or a different missing-value semantic, drop or recode the missing cells before passing the data in.
weighted	Logical. Apply exponential decay weighting to Hamming distance positions? Only valid when <code>dissimilarity = "hamming"</code> . Default: FALSE.
lambda	Numeric. Non-negative decay rate for weighted Hamming. Higher values weight earlier positions more strongly. Default: 1.
seed	Integer or NULL. Random seed for reproducibility. Default: NULL.
q	Integer. Size of q-grams for "qgram", "cosine", and "jaccard" distances. Default: 2L.
p	Numeric. Winkler prefix penalty for Jaro-Winkler distance. Must be between 0 and 0.25. Default: 0.1.
covariates	Optional. Post-hoc covariate analysis of cluster membership. Accepts: string Single column name, e.g. "Age". Resolved against <code>x\$metadata</code> (and <code>x\$data</code>) for <code>netobject</code> or <code>cograph_network</code> input. character vector <code>c("Age", "Gender")</code>, same lookup. formula <code>~ Age + Gender</code>, same lookup; supports "Age + Gender" string form too.

	data.frame All columns used as covariates verbatim; must have one row per sequence. NULL No covariate analysis (default). For <code>netobject</code> or <code>cograph_network</code> input, names are resolved against <code>\$metadata</code> first and then non-state columns of <code>\$data</code>, so a typical call looks like <code>build_clusters(net, k = 3, covariates = "session_label")</code> without pre-extracting a <code>data.frame</code>. <code>tna</code> input requires the <code>data.frame</code> form. Results are stored in <code>\$covariates</code>.
estimator	Multinomial logit fitter for the covariate analysis. "auto" (default) inspects the cluster x covariate cross-tab and falls back to "firth" only when any cell has fewer than 5 observations (quasi-complete separation risk); otherwise uses the much faster "multinom". "firth" forces Firth's penalised likelihood via <code>brglm2::brmultinom</code> – bias-reduced and finite under separation, but ~200x slower than <code>multinom</code> on well-conditioned data. "multinom" forces classical ML via <code>nnet::multinom</code> ; warns because rare-cell separation produces astronomical ORs with degenerate CIs (silent failure). "chisq" runs WeightedCluster-style descriptive tests (chi-square + Cramer's V + standardized adjusted residuals for factors; Kruskal-Wallis + eta-squared for numerics).
...	Unsupported. Supplying unused arguments raises an error. In <code>plot.net_clustering()</code> , <code>print.net_clustering()</code> and <code>summary.net_clustering()</code> : Unsupported. Supplying unused arguments raises an error. In <code>print.tidy_covariates()</code> : Ignored.
x	For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_clustering</code> or <code>tidy_covariates</code> .
digits	Integer. Decimal places used for floating-point statistics in the printout. Default 3. Non-breaking: existing <code>print(x)</code> calls keep their previous formatting.
object	For the <code>summary()</code> method: an object of class <code>net_clustering</code> .
type	Character. Plot type: "silhouette" (per-observation silhouette bars), "mds" (2D MDS projection), "heatmap" (distance matrix heatmap ordered by cluster), or "predictors" (odds-ratio forest plot of the post-hoc covariate analysis; requires covariates and an estimator that produces coefficients). Default: "silhouette".
combined	Logical. For <code>type = "predictors"</code> only: when TRUE (default), covariate forest panels are combined into a single faceted plot; when FALSE, a list of separate ggplots is returned.

Value

An object of class "net_clustering" containing:

data The original input data.

k Number of clusters.

assignments Named integer vector of cluster assignments.

silhouette Overall average silhouette width.

sizes Named integer vector of cluster sizes.

method Clustering method used.

dissimilarity Distance metric used.

distance The computed dissimilarity matrix (dist object).

medoids Integer vector of medoid row indices (PAM only; NULL for hierarchical methods).

seed Seed used (or NULL).

weighted Logical, whether weighted Hamming was used.

lambda Lambda value used (0 if not weighted).

covariates The post-hoc covariate analysis (a list; see the estimator argument), or NULL when covariates = NULL.

network_method, build_args For netobject input, the source network's method and stored build arguments, so per-cluster networks can be rebuilt the same way. NULL otherwise.

metadata For netobject input, its per-sequence metadata, one row per clustered sequence, so `session_ids` can name each sequence. NULL otherwise.

htna_partition For HTNA input, the preserved node-to-actor partition used to restore HTNA children when networks are built.

In `print.net_clustering()`: The input object, invisibly.

In `summary.net_clustering()`: A data frame of per-cluster statistics, one row per cluster, with columns `cluster`, `size` and `mean_within_dist`, returned *visibly*. When the clustering was fitted with covariates, a `tidy_covariates/data.frame` (the tidied covariate table, with cluster sizes, fit statistics and profiles attached as attributes) is returned *invisibly* instead. In both cases the printed summary is a side effect.

In `plot.net_clustering()`: A ggplot object (invisibly); for `type = "predictors"` with `combined = FALSE`, a list of ggplot objects named by cluster (invisibly).

In `print.tidy_covariates()`: The input invisibly.

Methods

- `print.net_clustering()`: Compact, fixed-width summary of a sequence-clustering result. The header carries the clustering method and dissimilarity; the per-cluster table carries cluster size (count and percentage) and mean within-cluster distance when available. Optional medoid and covariate lines surface only when those fields are populated.
- `print.tidy_covariates()`: Prints a one-line header naming the estimator, then the `data.frame`. The full human-readable view (per-cluster stats, profiles, OR/test tables) was already printed by `summary()` when this object was produced, so this method intentionally stays minimal to avoid duplication. Auto-prints when the user types the variable at the REPL.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A", "B"), V2 = c("B", "C", "A", "B", "A"),
                  V3 = c("C", "A", "B", "C", "B"))
cl <- build_clusters(seqs, k = 2)
cl

seqs <- data.frame(
  V1 = sample(LETTERS[1:3], 20, TRUE), V2 = sample(LETTERS[1:3], 20, TRUE),
  V3 = sample(LETTERS[1:3], 20, TRUE), V4 = sample(LETTERS[1:3], 20, TRUE)
```

```

)
cl <- build_clusters(seqs, k = 2)
print(cl)
summary(cl)

```

build_cna

*Build a Co-occurrence Network (CNA)***Description**

Convenience wrapper for `build_network(method = "co_occurrence")`. Computes co-occurrence counts from binary or sequence data.

Usage

```
build_cna(data, start = FALSE, end = FALSE, ...)
```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance). A fitted <code>net_clustering</code> or <code>net_mmm</code> object is also accepted: the per-cluster networks are (re)built and a <code>netobject_group</code> is returned.
start	Boundary marker prepended to every sequence as an explicit start state (a pure source: no incoming edges, every sequence's first transition is start -> first_observed). FALSE (default) adds nothing; TRUE uses the label "Start"; a single string uses that string as the label. Only valid for the transition methods (relative, frequency, co_occurrence, attention, ngram, gap, reverse); errors otherwise (wtna included).
end	Boundary marker placed in the single cell after each sequence's last observed (non-NA) state, as an explicit terminal state (a pure sink: no outgoing edges, no self-loop – distinct from <code>mark_terminal_state</code> , which fills all trailing NAs into an absorbing state). FALSE (default) adds nothing; TRUE uses the label "End"; a single string uses that string as the label. Same method restriction as start.
...	Additional arguments passed to <code>build_network</code> .

Value

A `netobject` (see `build_network`).

See Also

`build_network`, `cooccurrence` for delimited-field, bipartite, and other non-sequence co-occurrence formats.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
net <- build_cna(seqs)
```

build_cor

*Build a Correlation Network***Description**

Convenience wrapper for `build_network(method = "cor")`. Computes Pearson correlations from numeric data.

Usage

```
build_cor(data, ...)
```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance). A fitted <code>net_clustering</code> or <code>net_mmm</code> object is also accepted: the per-cluster networks are (re)built and a <code>netobject_group</code> is returned.
...	Additional arguments passed to build_network .

Value

A `netobject` (see [build_network](#)).

See Also

[build_network](#)

Examples

```
data(srl_strategies)
net <- build_cor(srl_strategies)
```

build_ftna

*Build a Frequency Transition Network (FTNA)***Description**

Convenience wrapper for `build_network(method = "frequency")`. Computes raw transition counts from sequence data.

Usage

```
build_ftna(data, start = FALSE, end = FALSE, ...)
```

Arguments

<code>data</code>	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance). A fitted <code>net_clustering</code> or <code>net_mmm</code> object is also accepted: the per-cluster networks are (re)built and a <code>netobject_group</code> is returned.
<code>start</code>	Boundary marker prepended to every sequence as an explicit start state (a pure source: no incoming edges, every sequence's first transition is <code>start -> first_observed</code>). <code>FALSE</code> (default) adds nothing; <code>TRUE</code> uses the label "Start"; a single string uses that string as the label. Only valid for the transition methods (<code>relative</code> , <code>frequency</code> , <code>co_occurrence</code> , <code>attention</code> , <code>ngram</code> , <code>gap</code> , <code>reverse</code>); errors otherwise (<code>wtna</code> included).
<code>end</code>	Boundary marker placed in the single cell after each sequence's last observed (non-NA) state, as an explicit terminal state (a pure sink: no outgoing edges, no self-loop – distinct from <code>mark_terminal_state</code> , which fills all trailing NAs into an absorbing state). <code>FALSE</code> (default) adds nothing; <code>TRUE</code> uses the label "End"; a single string uses that string as the label. Same method restriction as <code>start</code> .
<code>...</code>	Additional arguments passed to <code>build_network</code> .

Value

A `netobject` (see `build_network`).

See Also

[build_network](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
net <- build_ftna(seqs)
```

build_gimme

GIMME: Group Iterative Multiple Model Estimation

Description

Estimates person-specific directed networks from intensive longitudinal data using the unified Structural Equation Modeling (uSEM) framework. Implements a data-driven search that identifies:

1. **Group-level paths:** Directed edges present for a majority (default 75%) of individuals.
2. **Individual-level paths:** Additional edges specific to each person, found after group paths are established.

Estimation is delegated to `idiographic::fit_gimme()`, the clean-room home of the temporal idiographic estimators, whose search reproduces the upstream **gimme** package (≥ 10.0) exactly (verified at tolerance 0 on path counts and per-person coefficient matrices in idiographic's own parity suite). Uses lavaan for SEM estimation and modification indices. Accepts a single data frame with an ID column (not CSV directories).

Usage

```
build_gimme(
  data,
  vars,
  id,
  time = NULL,
  ar = TRUE,
  standardize = FALSE,
  groupcutoff = 0.75,
  subcutoff = 0.5,
  paths = NULL,
  exogenous = NULL,
  hybrid = FALSE,
  rmsea_cutoff = 0.05,
  srmr_cutoff = 0.05,
  nnfi_cutoff = 0.95,
  cfi_cutoff = 0.95,
  n_excellent = 2L,
  seed = NULL
)
```

Arguments

data	A data.frame in long format with columns for person ID, time-varying variables, and optionally a time/beep column.
vars	Character vector of variable names to model.
id	Character string naming the person-ID column.

time	Character string naming the time/order column, or NULL. When provided, data is sorted by id then time before lagging.
ar	Logical. If TRUE (default), autoregressive paths (each variable predicting itself at lag 1) are included as fixed paths.
standardize	Logical. If TRUE (default FALSE), variables are standardized per person before estimation.
groupcutoff	Numeric between 0 and 1. Proportion of individuals for whom a path must be significant to be added at group level. Default 0.75.
subcutoff	Numeric. Not used (reserved for future subgrouping); accepted for API compatibility and <i>not</i> forwarded to <code>idiographic::fit_gimme()</code> , which does not implement subgrouping either. Default 0.50.
paths	Character vector of lavaan-syntax paths to force into the model (e.g., "V2~V1lag"). Default NULL.
exogenous	Character vector of variable names to treat as exogenous. Default NULL.
hybrid	Logical. If TRUE, also searches residual covariances. Default FALSE.
rmsea_cutoff	Numeric. RMSEA threshold for excellent fit (default 0.05).
srmr_cutoff	Numeric. SRMR threshold for excellent fit (default 0.05).
nnfi_cutoff	Numeric. NNFI/TLI threshold for excellent fit (default 0.95).
cfi_cutoff	Numeric. CFI threshold for excellent fit (default 0.95).
n_excellent	Integer. Number of fit indices that must be excellent to stop individual search. Default 2.
seed	Integer or NULL. Random seed for reproducibility.

Value

The object returned by `idiographic::fit_gimme()`: an S3 object of class `c("net_gimme", "cograph_network", "list")`. It is a *superset* of the pre-0.9.0 in-package field contract – every element below is present, alongside idiographic’s own additions (`contemp_cov`, `contemp_cov_avg`, `contemp_is_cov`). Elements:

temporal $p \times p$ matrix of group-level temporal (lagged) path counts – entry $[i, j]$ = number of individuals with path $j(t-1) \rightarrow i(t)$.

contemporaneous $p \times p$ matrix of group-level contemporaneous path counts – entry $[i, j]$ = number of individuals with path $j(t) \rightarrow i(t)$.

temporal_avg, **contemporaneous_avg** $p \times p$ group-average coefficient matrices.

coefs List of per-person $p \times 2p$ coefficient matrices (rows = endogenous, cols = [lagged, contemporaneous]).

psi List of per-person residual covariance matrices.

fit Data frame of per-person fit indices (`chisq`, `df`, `pvalue`, `rmsea`, `srmr`, `nnfi`, `cfi`, `bic`, `aic`, `logl`, `status`).

path_counts $p \times 2p$ matrix: how many individuals have each path.

paths List of per-person character vectors of lavaan path syntax.

group_paths Character vector of group-level paths found.

individual_paths List of per-person character vectors of individual-level paths (beyond group).
 syntax List of per-person full lavaan syntax strings.
 labels Character vector of variable names.
 n_subjects Integer. Number of individuals.
 n_obs Integer vector. Time points per individual.
 config List of configuration parameters.

The object additionally carries idiographic's netobject fields (weights, nodes, edges, directed, data, meta, node_groups) so it renders directly with cograph. `print()`, `summary()` and `plot()` dispatch to **idiographic**'s methods, not to Nestimate's: in particular `summary()` returns a tidy `data.frame` rather than printing.

Results changed in 0.9.0

Before 0.9.0 the search ran in an in-package implementation that was *not* upstream-gimme-exact. Delegating to `idiographic::fit_gimme()` changed which paths the search selects on the same data – individual-level paths in particular – so numeric results are not comparable with Nestimate $\leq 0.8.5$. The returned object also gained fields (see **Value**); nothing was removed.

See Also

[build_network](#)

Examples

```
# Create simple panel data (3 subjects, 4 variables, 30 time points).
set.seed(42)
n_sub <- 3; n_t <- 30; vars <- paste0("V", 1:4)
rows <- lapply(seq_len(n_sub), function(i) {
  d <- as.data.frame(matrix(rnorm(n_t * 4), ncol = 4))
  names(d) <- vars; d$id <- i; d
})
panel <- do.call(rbind, rows)
res <- build_gimme(panel, vars = vars, id = "id")
print(res)
```

Description

Convenience wrapper for `build_network(method = "glasso")`. Computes L1-regularized partial correlations with EBIC model selection.

Usage

```
build_glasso(data, ...)
```

Arguments

<code>data</code>	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance). A fitted <code>net_clustering</code> or <code>net_mmm</code> object is also accepted: the per-cluster networks are (re)built and a <code>netobject_group</code> is returned.
<code>...</code>	Additional arguments passed to build_network .

Value

A `netobject` (see [build_network](#)).

See Also

[build_network](#)

Examples

```
data(srl_strategies)
net <- build_glasso(srl_strategies)
```

build_hon

Build a Higher-Order Network (HON)

Description

Constructs a Higher-Order Network from sequential data, faithfully implementing the BuildHON algorithm (Xu, Wickramaratne & Chawla, 2016).

The algorithm detects when a first-order Markov model is insufficient to capture sequential dependencies and automatically creates higher-order nodes. Uses KL-divergence to determine whether extending a node's history provides significantly different transition distributions.

Usage

```
build_hon(
  data,
  max_order = 5L,
  min_freq = 1L,
  collapse_repeats = FALSE,
  method = "hon+"
)

## S3 method for class 'net_hon'
print(x, ...)
```

```
## S3 method for class 'net_hon'
summary(object, ...)
```

Arguments

data	One of: • data.frame: rows are trajectories, columns are time steps. Trailing NAs are stripped. All non-NA values are coerced to character. • list: each element is a character (or coercible) vector representing one trajectory. • tna: a tna object with sequence data. Numeric state IDs are automatically converted to label names. • netobject: a netobject with sequence data.
max_order	Integer. Maximum order of the HON. Default 5. The algorithm may produce lower-order nodes if the data do not justify higher orders.
min_freq	Integer. Minimum frequency for a transition to be considered. Transitions observed fewer than min_freq times are treated as zero. Default 1.
collapse_repeats	Logical. If TRUE, adjacent duplicate states within each trajectory are collapsed before analysis. Default FALSE.
method	Character. Algorithm to use: "hon+" (default, parameter-free BuildHON+ with lazy observation building and MaxDivergence pruning) or "hon" (original BuildHON with eager observation building).
x	For the print() method: an object of class net_hon.
...	In print.net_hon() and summary.net_hon(): Additional arguments (ignored).
object	For the summary() method: an object of class net_hon.

Details

Node naming convention: Higher-order nodes use readable arrow notation. A first-order node is simply "A". A second-order node representing the context "came from A, now at B" is "A -> B". Third-order: "A -> B -> C", etc.

Algorithm overview:

1. Count all subsequence transitions up to max_order + 1.
2. Build probability distributions, filtering by min_freq.
3. For each first-order source, recursively test whether extending the history (adding more context) produces a significantly different distribution (via KL-divergence vs. an adaptive threshold).
4. Build the network from the accepted rules, rewiring edges so higher-order nodes are properly connected.

Value

An S3 object of class `c("net_hon", "cograph_network")` containing:

weights, matrix The same weighted adjacency matrix (rows = from, cols = to) under both names; `weights` is the `cograph_network` slot, `matrix` the higher-order name kept for back-compatibility. Rows and columns use readable arrow notation (e.g., "A -> B").

ho_edges The higher-order edge table: one row per HON edge, with columns `path` (full state sequence, e.g., "A -> B -> C"), `from` (context/conditioning states), `to` (predicted next state), `count` (raw frequency), `probability` (transition probability), `from_order`, `to_order`.

edges The `cograph_network` edge table: one row per non-zero cell of `weights`, with *integer* `from/to` node indices into nodes and a numeric weight. This is **not** the arrow-notation table - use `ho_edges` for that.

nodes `data.frame` with columns `id`, `label`, `name` (one row per HON node; `label/name` are the arrow-notation node names). Stored as a `data.frame` for `cograph_network` compatibility.

n_nodes Number of HON nodes.

n_edges Number of rows in `ho_edges`.

first_order_states Character vector of unique original states.

max_order_requested The `max_order` parameter used.

max_order_observed Highest order actually present.

min_freq The `min_freq` parameter used.

n_trajectories Number of trajectories after parsing.

directed Logical. Always TRUE.

meta `cograph_network` metadata list (`source`, `layout`, `tna$method` = "hon").

node_groups Always NULL.

In `print.net_hon()`: The input object, invisibly.

In `summary.net_hon()`: The `cograph_network` edge `data.frame` object `$edges`: one row per non-zero cell of the adjacency matrix, with integer `from/to` node indices and a numeric weight. Returned visibly; the summary text (counts, first-order states, order distribution) is printed as a side effect. The arrow-notation table with `path/count/probability` is object `$ho_edges`.

References

Xu, J., Wickramaratne, T. L., & Chawla, N. V. (2016). Representing higher-order dependencies in networks. *Science Advances*, 2(5), e1600028.

Saebi, M., Xu, J., Kaplan, L. M., Ribeiro, B., & Chawla, N. V. (2020). Efficient modeling of higher-order dependencies in networks: from algorithm to application for anomaly detection. *EPJ Data Science*, 9(1), 15.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hon <- build_hon(seqs, max_order = 2)
```

```
# From list of trajectories
trajs <- list(
  c("A", "B", "C", "D", "A"),
  c("A", "B", "D", "C", "A"),
  c("A", "B", "C", "D", "A")
)
hon <- build_hon(trajs, max_order = 3, min_freq = 1)
print(hon)
summary(hon)

# From data.frame (rows = trajectories)
df <- data.frame(T1 = c("A", "A"), T2 = c("B", "B"),
                 T3 = c("C", "D"), T4 = c("D", "C"))
hon <- build_hon(df, max_order = 2)
```

build_honem

Build HONEM Embeddings for Higher-Order Networks

Description

Constructs low-dimensional embeddings from a Higher-Order Network (HON) that preserve higher-order dependencies. Uses exponentially-decaying matrix powers of the HON transition matrix followed by truncated SVD.

Usage

```
build_honem(hon, dim = 32L, max_power = 10L)
```

```
## S3 method for class 'net_honem'
print(x, ...)
```

```
## S3 method for class 'net_honem'
summary(object, ...)
```

```
## S3 method for class 'net_honem'
plot(x, dims = c(1L, 2L), ...)
```

Arguments

hon	A net_hon object from build_hon , or a square weighted adjacency matrix.
dim	Integer. Embedding dimension (default 32). Silently capped at <code>n_nodes - 1</code> ; the dimension actually used is reported in the returned <code>dim</code> component.
max_power	Integer. Maximum walk length for neighborhood computation (default 10). Higher values capture longer-range structure.
x	For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_honem</code> .

...	In <code>plot.net_honem()</code> : Additional arguments passed to <code>plot</code> . In <code>print.net_honem()</code> and <code>summary.net_honem()</code> : Additional arguments (ignored).
object	For the <code>summary()</code> method: an object of class <code>net_honem</code> .
dims	Integer vector of length 2. Dimensions to plot (default: <code>c(1, 2)</code>).

Details

HONEM is parameter-free and scalable - no random walks, skip-gram, or hyperparameter tuning required.

Value

An object of class `net_honem` with components:

embeddings Numeric matrix (`n_nodes` x `dim`) of node embeddings, row names = node names, column names `dim_1`, `dim_2`, ...

nodes Character vector of node names.

singular_values Numeric vector of top singular values.

explained_variance Proportion of variance explained.

dim Embedding dimension used.

max_power Maximum power used.

n_nodes Number of nodes embedded.

In `print.net_honem()` and `plot.net_honem()`: The input object, invisibly.

In `summary.net_honem()`: A data.frame with one row per node: column `node` (node label) followed by `dim1`, `dim2`, ..., `dimd` embedding coordinates, returned visibly; the summary text is printed as a side effect.

References

Saebi, M., Ciampaglia, G. L., Kaplan, L. M., & Chawla, N. V. (2020). HONEM: Learning Embedding for Higher Order Networks. *Big Data*, 8(4), 255-269.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hem <- build_honem(build_hon(seqs, max_order = 2), dim = 2)
```

```
trajs <- list(c("A","B","C","D"), c("A","B","D","C"),
             c("B","C","D","A"), c("C","D","A","B"))
hon <- build_hon(trajs, max_order = 2)
emb <- build_honem(hon, dim = 4)
print(emb)
plot(emb)
```

build_hypa

Detect Path Anomalies via HYPA

Description

Constructs a k -th order De Bruijn graph from sequential trajectory data and uses a hypergeometric null model to detect paths with anomalous frequencies. Paths occurring more or less often than expected under the null model are flagged as over- or under-represented.

Usage

```
build_hypa(
  data,
  order = 2L,
  alpha = 0.05,
  min_count = 5L,
  p_adjust = "BH",
  k = NULL
)

## S3 method for class 'net_hypa'
print(x, ...)

## S3 method for class 'net_hypa'
summary(
  object,
  n = 10L,
  type = c("all", "over", "under"),
  order_by = c("sig", "freq", "frequency", "ratio", "path"),
  ...
)
```

Arguments

data	A data.frame (rows = trajectories), list of character vectors, tna object, or netobject with sequence data. For tna/netobject, numeric state IDs are automatically converted to label names.
order	Integer scalar or integer vector. Order(s) of the De Bruijn graph (default 2L). An order of k detects anomalies in paths of length k . When a vector is supplied, one De Bruijn layer is built per order and the per-order results are stored in \$by_order (named by order). The orders are sorted ascending internally, so the cograph_network slots (\$weights, \$edges, \$adjacency, \$xi, \$nodes, \$meta) always describe the network of the <i>lowest order that produced a layer</i> (a requested order with no edges is dropped from \$by_order, \$order and \$k), regardless of the order in which the vector is given; \$scores aggregates every built order.

<code>alpha</code>	Numeric in (0, 0.5). Significance threshold for anomaly classification (default 0.05). A path is labelled "under" when its adjusted lower-tail p-value <code>p_adjusted_under</code> is below <code>alpha</code> , "over" when its adjusted upper-tail p-value <code>p_adjusted_over</code> is below <code>alpha</code> , and "normal" otherwise (the under-representation test is applied first).
<code>min_count</code>	Integer. Minimum observed count for a path to be classified as anomalous (default 5). Paths with fewer observations are always classified as "normal" regardless of their HYPA score, since rare occurrences are unreliable.
<code>p_adjust</code>	Character. Method for multiple testing correction of p-values. Default "BH" (Benjamini-Hochberg FDR control). Accepts any method from p.adjust.methods or "none" to skip correction. Under- and over-representation p-values are adjusted separately (two-sided testing).
<code>k</code>	Deprecated. Former name of <code>order</code> ; if supplied it overrides <code>order</code> and emits a deprecation message. Use <code>order</code> instead.
<code>x</code>	For the <code>print()</code> method: an object of class <code>net_hypa</code> .
<code>...</code>	In <code>print.net_hypa()</code> and <code>summary.net_hypa()</code> : Additional arguments (ignored).
<code>object</code>	For the <code>summary()</code> method: an object of class <code>net_hypa</code> .
<code>n</code>	Integer. Maximum number of paths to display per category (default: 10).
<code>type</code>	Character. Which anomalies to show: "all" (default), "over", or "under".
<code>order_by</code>	Character. Ranking used within each anomaly direction: "sig" (default) ranks by the active tail probability, "freq" (or its alias "frequency") by observed count, "ratio" by observed/expected ratio, and "path" alphabetically.

Value

An object of class `c("net_hypa", "cograph_network")` with components:

scores Data frame with `path`, `from`, `to`, `observed`, `expected`, `ratio`, `p_value`, `p_under`, `p_over`, `p_adjusted_under`, `p_adjusted_over`, `anomaly`, `order` columns (one block of rows per requested order). The `path` column shows the full state sequence (e.g., "A -> B -> C"); `from` is the context (conditioning states); `to` is the next state; `ratio` is `observed / expected`; `p_value` is retained as an alias for `p_under`, the raw lower-tail hypergeometric CDF value; `p_over` is the inclusive upper-tail probability $P(X \geq \text{observed})$; `p_adjusted_under` and `p_adjusted_over` are the corrected p-values for under- and over-representation tests respectively.

ho_edges Alias for `scores` (all orders, arrow notation).

over Subset of `scores` classified as over-represented.

under Subset of `scores` classified as under-represented.

adjacency Weighted adjacency matrix of the lowest-order De Bruijn graph.

weights cograph weight matrix of the lowest-order graph.

xi Fitted propensity matrix of the lowest-order graph.

edges cograph edge data.frame of the lowest-order graph.

by_order Named list of per-order result lists.

order Integer vector of orders actually built (sorted ascending).

k Back-compatibility alias for order.

alpha Significance threshold used.

p_adjust Multiple testing correction method used.

n_anomalous Number of anomalous paths detected (all orders).

n_over Number of over-represented paths (all orders).

n_under Number of under-represented paths (all orders).

n_edges Total number of edges (all orders).

nodes data.frame (id, label, name) of the lowest-order De Bruijn graph nodes (arrow notation).

directed Logical. Always TRUE.

meta cograph meta list of the lowest-order graph.

node_groups Always NULL.

In `print.net_hypa()`: The input object, invisibly.

In `summary.net_hypa()`: A data frame of the reported anomalies, at most n rows per direction, with columns order (the De Bruijn order the path was found at), path, observed, expected, ratio, p_tail (the raw tail probability in the reported direction: p_over for over-represented paths, p_under for under-represented ones) and direction ("over"/"under"). Returned visibly; the summary text and the top-n tables are printed as a side effect. When no anomalies were detected, a zero-row data frame with the same columns except order is returned.

References

LaRock, T., Nanumyan, V., Scholtes, I., Casiraghi, G., Eliassi-Rad, T., & Schweitzer, F. (2020). HYPA: Efficient Detection of Path Anomalies in Time Series Data on Networks. *SDM 2020*, 460-468.

Examples

```
seqs <- list(c("A","B","C"), c("B","C","A"), c("A","C","B"), c("A","B","C"))
hyp <- build_hypa(seqs, order = 2)
```

```
trajs <- list(c("A","B","C"), c("A","B","C"), c("A","B","C"),
             c("A","B","D"), c("C","B","D"), c("C","B","A"))
h <- build_hypa(trajs, order = 2)
print(h)
```

build_hypergraph	<i>Higher-order hypergraph from a network's clique structure</i>
------------------	--

Description

Takes a network and produces a hypergraph by promoting k -cliques ($k \geq 3$) to k -hyperedges. Each k -clique is independently included as a k -hyperedge with probability p . Optionally retains the underlying pairwise edges as 2-hyperedges. Foundation for higher-order analyses.

Usage

```
build_hypergraph(
  net,
  p = 1,
  method = c("clique", "vr", "rips"),
  include_pairwise = TRUE,
  max_size = 3L,
  threshold = 0,
  seed = NULL
)

## S3 method for class 'net_hypergraph'
print(x, ...)

## S3 method for class 'net_hypergraph'
summary(object, ...)
```

Arguments

net	A netobject, cograph_network, simplicial_complex, or numeric adjacency / weight matrix. Directed inputs are symmetrised by the underlying clique enumerator.
p	Probability in $[0, 1]$ that each k -clique with $k \geq 3$ becomes a k -hyperedge. Default 1 (deterministic - every found clique is included).
method	Hyperedge enumeration. "clique" (default) promotes k -cliques in the binarised adjacency to k -hyperedges. A metric Vietoris-Rips construction is not implemented ; "vr" / "rips" are accepted by <code>match.arg</code> but raise an error rather than silently aliasing "clique".
include_pairwise	Logical. Include 2-edges from the input network as 2-hyperedges. Default TRUE. Set FALSE for a "fully higher-order" hypergraph containing only k -hyperedges with $k \geq 3$.
max_size	Integer ≥ 2 . Maximum hyperedge size to extract. Default 3L (triangles only). 4L also includes 4-cliques as 4-hyperedges, etc.
threshold	Numeric. Edge weight cutoff used to binarise the adjacency for clique enumeration. Default 0 (any non-zero weight is an edge).

seed	Optional integer for reproducible Bernoulli sampling when $0 < p < 1$.
x	For the <code>print()</code> method: an object of class <code>net_hypergraph</code> .
...	In <code>print.net_hypergraph()</code> : Additional arguments (ignored). In <code>summary.net_hypergraph()</code> : Ignored.
object	For the <code>summary()</code> method: an object of class <code>net_hypergraph</code> .

Details

The construction follows Burgio, Matamalas, Gomez & Arenas (2020) on simplicial / hypergraph contagion. For each k -clique with $k \geq 3$ found in the underlying graph (via `build_simplicial()`), an independent Bernoulli(p) trial decides whether that clique becomes a k -hyperedge. Underlying pairwise edges are always retained when `include_pairwise = TRUE`, so the resulting hypergraph contains both the original 2-edge structure and the sampled higher-order interactions.

At the limits:

- $p = 0$ with `include_pairwise = TRUE` reproduces the input pairwise network as a hypergraph of size-2 edges.
- $p = 1$ with `include_pairwise = FALSE` returns a fully higher-order hypergraph containing only the k -hyperedges ($k \geq 3$) found in the network's clique complex.

Value

A `net_hypergraph` object: a list with components

`hyperedges` List of integer vectors. Each entry is a hyperedge given as the sorted node indices it spans.

`incidence` Numeric matrix of size `n_nodes` x `n_hyperedges`. `incidence[i, j] = 1` iff node i belongs to hyperedge j . Row names are node names; column names are `h1, h2, ...`

`nodes` Character vector of node names.

`n_nodes, n_hyperedges` Scalar counts.

`size_distribution` Named integer vector: number of hyperedges of each size, named `size_2, size_3, ...`

`params` Recorded call parameters: `method, p, include_pairwise, max_size, threshold, seed`.

In `print.net_hypergraph()`: For `print()`, the input `x` invisibly.

In `summary.net_hypergraph()`: For `summary()`, a `data.frame` with one row per node and columns `node` (node name) and `degree` (number of hyperedges containing the node), returned visibly; the summary block (node / hyperedge counts, mean and maximum hyperedge size) is printed as a side effect.

References

Burgio, G., Matamalas, J. T., Gomez, S., & Arenas, A. (2020). Evolution of cooperation in the presence of higher-order interactions: from networks to hypergraphs. *Entropy* 22(7), 744. doi:10.3390/e22070744

See Also

[build_simplicial\(\)](#) (underlying clique enumeration), [build_network\(\)](#).

Examples

```
set.seed(1)
n <- 8
adj <- matrix(stats::rbinom(n * n, 1, 0.5), n, n)
diag(adj) <- 0
adj <- (adj + t(adj)) > 0
rownames(adj) <- colnames(adj) <- LETTERS[seq_len(n)]
hg <- build_hypergraph(adj, p = 1, max_size = 3L)
print(hg)
summary(hg)
```

 build_ising

Build an Ising Network

Description

Convenience wrapper for `build_network(method = "ising")`. Computes L1-regularized logistic regression network for binary data.

Usage

```
build_ising(data, ...)
```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance). A fitted <code>net_clustering</code> or <code>net_mmm</code> object is also accepted: the per-cluster networks are (re)built and a <code>netobject_group</code> is returned.
...	Additional arguments passed to build_network .

Value

A `netobject` (see [build_network](#)).

See Also

[build_network](#)

Examples

```
if (requireNamespace("glmnet", quietly = TRUE)) {
  bin_data <- data.frame(matrix(rbinom(200, 1, 0.5), ncol = 5))
  net <- build_ising(bin_data)
}
```


	data.frame with from/to columns Edge list. Columns named from/source/src/v1/node1/i and to/target/tgt/v2/node2/j are auto-detected. Optional weight column (weight/w/value/strength). data.frame without from/to columns Sequence data. Each row is a sequence, columns are time steps. Consecutive pairs (t, t+1) become transitions. tna object If x\$data is non-NULL, uses sequence path on the raw data. Otherwise falls back to cluster_summary. netobject If x\$data is non-NULL, detects edge list vs sequence data. Otherwise falls back to cluster_summary. mcml An mcml object is returned unchanged, unless clusters, combine or expand changes its partition: it is then re-estimated from the sequences it carries (see combine). square numeric matrix Falls back to cluster_summary. non-square or character matrix Treated as sequence data. For the print() method: an object of class mcml_layer or mcml.
clusters	Cluster/group assignments. Accepts: named list Direct mapping. List names = cluster names, values = character vectors of node labels. Example: list(A = c("N1", "N2"), B = c("N3", "N4")) data.frame A data frame where the first column contains node names and the second column contains group/cluster names. Example: data.frame(node = c("N1", "N2", "N3"), group = c("A", "A", "B")) membership vector Character or numeric vector. Node names are extracted from the data. Example: c("A", "A", "B", "B") column name string For edge list data.frames, the name of a column containing cluster labels. The mapping is built from unique (node, group) pairs in both from and to columns. Limitation: this mode assigns the row's group label to <i>both</i> endpoints, so it only makes sense for edge lists where source and target nodes always share the same group (within-group edges only). For general edge lists where a single node may be source in some rows and target in others, or where source and target belong to different groups, pass an explicit named list (list(G1 = c("N1", "N2"), ...)) or a two-column data frame data.frame(node, group) instead. NULL Auto-detect from a netobject's node table (a clusters/cluster/groups/group column) or from its node_groups. Only netobject input can auto-detect; data frames and matrices require an explicit assignment.
method	Aggregation method for combining edge weights: "sum", "mean", "median", "max", "min", "density", "geomean". Default "sum". For raw sequence/event-log inputs the function is counting observed transitions, so "sum" is the only interpretation that preserves the count semantics – the other methods are useful when aggregating weighted edge lists or pre-existing weight matrices, where each row already represents a measurement rather than a single observation.
type	Post-processing of the aggregated count matrix. One of: "tna" (default) Row-normalize so each row sums to 1 (first-order Markov transition probabilities). "raw" Return the un-normalized count matrix.

	"frequency" Explicit alias of "raw" – identical raw count construction (kept as a synonym for callers using frequency-network terminology). "cooccurrence" Symmetrize the matrix (undirected co-occurrence). "semi_markov" is <i>not</i> accepted: the package does not implement a semi-Markov / holding-time construction, so passing it errors rather than silently aliasing "tna".
directed	Logical. If TRUE (default), treat transitions as directed. If FALSE, symmetrize sequence- and edge-derived weights before returning raw/frequency weights or before row-normalizing transition probabilities.
compute_within	Logical. Compute within-cluster matrices? Default TRUE.
actor, action, time, order, session, time_threshold	Long-format event-log shortcut. When action is supplied on a data.frame input, the data is passed through prepare() to derive a wide sequence, which is then routed to the existing sequence path. Behaves identically to prepare(...) > build_network() > build_mcml().
exclude	Optional character vector of state labels to drop before the network is built (e.g. a technical-void marker). On long-format input the matching events are removed before sequences are formed, so a dropped state never occupies a sequence position; on wide sequence input the matching cells are set to NA.
trim	Optional truncation of each sequence. NULL (default) keeps every time point. A fraction in (0, 1) keeps the columns covering that quantile of sequence lengths (trim = 0.95 keeps the shortest 95%); a value >= 1 is an absolute cut (trim = 10 keeps the first 10 time points). Same semantics as sequence_plot 's trim.
end	Terminal state appended after each sequence's last observed state. FALSE (default) adds none, TRUE adds one labelled "End", and a string supplies the label. Applied <i>after</i> trim, so trimming can never remove the marker.
end_by	Optional column name(s) grouping the terminal marker at a coarser unit than the sequence. NULL (default) marks every sequence. When supplied, only the <i>last</i> sequence of each group is marked – e.g. session = "AttemptID", end = "Conclude", end_by = "SkillID" closes each skill once, not each attempt. Requires long-format input, since the grouping column lives there.
labels	Optional name -> label remap applied to within-cluster nodes (the macro layer is left untouched because its labels are cluster names). Accepts a 2-column data.frame (name, label), a named character vector c(name = "label"), or a named list. Unmapped names pass through unchanged.
combine, expand	Change the partition. On new input they apply to clusters (in any of its accepted forms, including auto-detection) before estimation, so build_mcml(data, clusters = cl, combine = c("A", "B")) equals a build with A and B merged in cl (the merged cluster lists its states in cluster-name order); this works for every input type, matrices included. On an existing mcml they re-partition it (see below). combine merges clusters into one: a character vector merges one group, a list merges several and its names label them (default label "A + B"). expand then splits the named clusters (or "all"/TRUE) into one cluster per member state, named by the state. For an existing mcml the model is re-estimated – macro network, within-cluster networks and stored sequences – from the sequences the

mcml carries, with its original type, method and directed unless passed explicitly. Any session split, exclude, trim or end applied when it was built is already in those sequences, so passing any of these (or actor, action, time, session, labels) together with a re-partition is an error. Passing a new clusters list instead re-estimates under that partition; with the partition unchanged the result equals the input. Errors with class `nestimate_mcml_no_sequences` when the mcml was built from a matrix, from an edge list (only within-cluster edges are kept), or with `compute_within = FALSE`. `sequence_plot` accepts the same two arguments as `display_options`: its `combine` draws exactly what it draws for `build_mcml(x, combine = )`, while its `expand` opens clusters in the Summary panel only and keeps one panel per cluster.

... In `print.mcml()`, `print.mcml_layer()` and `summary.mcml()`: Unsupported. Supplying unused arguments raises an error.

object For the `summary()` method: an object of class `mcml`.

Value

An `mcml` object with the same layout as the return value of `cluster_summary` (macro, clusters, cluster_members, edges, meta). On the sequence and edge-list paths `meta$source` is "transitions", `meta$type` records the type post-processing, and `edges` is a tidy data frame with one row per observed node-level transition and columns `from`, `to`, `weight`, `cluster_from`, `cluster_to`, `type` ("within"/"between"). Matrix input falls through to `cluster_summary`, so `meta$source` is "matrix" and `edges` is NULL. Works with `print()`, `summary()`, `as_tna`, `as_htna` and `macro_network`.

In `print.mcml_layer()`: The input `mcml_layer`, invisibly.

In `print.mcml()`: The input object, invisibly.

In `summary.mcml()`: A tidy data frame with one row per cluster and columns `cluster`, `size`, `within_total`, `between_out`, `between_in`. For undirected macro networks the in/out split is not meaningful, so `between_out` reports total incident weight and `between_in` is NA. The data frame is returned silently *without* printing the full object – call `print(object)` explicitly if you want the verbose dump.

Methods

- `print.mcml_layer()`: Compact view of one `mcml` layer (the macro layer or a single within-cluster network): a header line with the node and non-zero edge counts and the weight range, the rounded weight matrix, the initial probabilities as a bar chart, and the dimensions of any attached data – rather than the raw list contents.

See Also

`cluster_summary` for matrix-based aggregation, `as_tna` to promote the layers to `netobjects`, `macro_network` for the cluster-level network with one cluster expanded

Examples

```
# Edge list with clusters
edges <- data.frame(
  from = c("A", "A", "B", "C", "C", "D"),
```

```

    to = c("B", "C", "A", "D", "D", "A"),
    weight = c(1, 2, 1, 3, 1, 2)
  )
clusters <- list(G1 = c("A", "B"), G2 = c("C", "D"))
build_mcml(edges, clusters)

# Sequence data with clusters
seqs <- data.frame(
  T1 = c("A", "C", "B"),
  T2 = c("B", "D", "A"),
  T3 = c("C", "C", "D"),
  T4 = c("D", "A", "C")
)
cs <- build_mcml(seqs, clusters, type = "raw")
cs
summary(cs)

# Change the partition while building ...
three <- build_mcml(seqs, list(G1 = "A", G2 = "B", G3 = c("C", "D")))
build_mcml(seqs, list(G1 = "A", G2 = "B", G3 = c("C", "D")),
  combine = c("G1", "G2"))

# ... or re-partition an existing mcml: the model is re-estimated
build_mcml(three, combine = c("G1", "G2")) # G1 + G2 as one cluster
build_mcml(three, combine = list(AB = c("G1", "G2"))) # named merge
build_mcml(three, expand = "G3") # C and D as clusters

```

build_mcml_pc

Multi-Cluster Multi-Level Aggregation for Psychometric Networks

Description

Experimental. Aggregates a node-level psychometric network (correlation, partial correlation, or EBICglasso) into a cluster-level macro network plus per-cluster within networks - the MCML view that `build_mcml` provides for transition networks, adapted to the statistics of undirected association networks. The API and the exact aggregation formulas may change between releases.

Unlike transition counts, partial correlations do not aggregate by arithmetic: the submatrix of a pcor matrix is *not* the pcor network of the subsystem (the conditioning set changes), and averaging pcor entries across blocks is descriptive only. The aggregation methods therefore have explicitly different statuses (the full list of accepted values is under aggregation below):

"average" (**descriptive**) Macro edge A–B = mean of the signed node-level weights between members of A and members of B; macro diagonal = mean within-block off-diagonal weight. Needs only the weight matrix (works on data-less netobjects). Caveat: signed averaging can cancel opposite-sign edges; interpret as net average association, not connectivity strength.

"composite" (**re-estimated**) Per-observation cluster scores = (optionally standardized) mean of member variables; the chosen method is then re-fit on the k composite columns, so the macro network is a genuine correlation / pcor / EBICglasso network among clusters. Requires raw data.

"loadings" (**re-estimated, connectivity-weighted**) As "composite", but member variables are weighted by their within-cluster connection strength in the node-level network (the absolute summed weight to the other members of their own cluster, normalized to sum to 1 per cluster). Nodes that anchor their cluster contribute more to its composite. This is Nestimate's own weighting - related in spirit to network loadings (Christensen & Golino 2021) but not a reimplementaion of any EGA-family estimator. Requires raw data.

"escoufier" (**descriptive, multivariate**) Macro edge A–B = Escoufier's RV coefficient between the member blocks - a matrix correlation in $[0, 1]$ computed from the block covariance structure. No composites are formed and no estimator is re-fit, so nothing is lost to averaging; signs are not represented. Requires raw data.

"cancor" (**descriptive, multivariate**) Macro edge A–B = the first canonical correlation between the member blocks: the strongest linear relationship any weighting of A's items can have with any weighting of B's items - an upper bound on what composite methods can recover. Requires raw data.

Usage

```
build_mcml_pc(
  x,
  clusters,
  aggregation = c("scaled", "composite", "mean", "median", "loadings", "average",
    "escoufier", "cancor"),
  method = c("pcor", "glasso", "cor"),
  within = c("reestimate", "subnetwork"),
  weighting = c("equal", "strength", "eigen", "closeness", "betweenness",
    "expected_influence", "specificity", "pca", "factor", "item_total"),
  cor_method = c("pearson", "spearman", "polychoric"),
  signed = TRUE,
  id_col = NULL,
  fa_method = c("ml", "paf", "minres", "cfa"),
  ...
)

## S3 method for class 'mcml_pc'
print(x, digits = 3, ...)

## S3 method for class 'mcml_pc'
summary(object, ...)

## S3 method for class 'mcml_pc'
plot(x, digits = 2, ...)
```

Arguments

x A netobject estimated with an undirected association method (cor, pcors, glasso; aliases accepted) - its \$data and \$method are reused - or a numeric data.frame of raw observations (then method decides the estimator). For the print() and plot() methods: an object of class mcml_pc.

clusters	Cluster membership in any of three forms: a named list of node-label vectors (names = cluster labels); a two-column data.frame read by position (first column = node names, second = group labels); or a vector of cluster labels named by node. Every node must be assigned to exactly one cluster.
aggregation	Character. How clusters are collapsed to the macro network. Default "scaled". "scaled" Re-estimate the network on per-cluster scores formed as the mean of <i>standardized</i> member items (the default; was "composite"). "composite" Backward-compatible alias for "scaled". "mean" As "scaled" but on raw (unstandardized) item means. "median" As "mean" but the per-cluster score is the row-wise median of the items (unweighted). "loadings" The scaled score path with member items weighted by their within-cluster network strength. "average" Average the item-pair edges in each between-cluster submatrix - no scores, no re-estimation. "escoufier" Escoufier RV coefficient (descriptive multivariate similarity between blocks). "cancor" First canonical correlation - an upper bound on how related two blocks can be.
method	Character. Network estimator for the re-estimation paths and for data.frame input: "pcor" (default), "glasso", or "cor" - the same vocabulary as build_network. Ignored (with the netobject's own method used instead) when x is a netobject and method is not given.
within	Character. "reestimate" (default) or "subnetwork" (see Details).
weighting	Character. How items are weighted inside their cluster score (the score paths "scaled" / "mean" / "loadings"; not "median", which is unweighted): "equal" (default) 1/m per item - the scale as scored. "strength" Mean absolute connection to the other own-cluster members in the node-level network (what aggregation = "loadings" selects). "eigen" Leading-eigenvector weights of the within-cluster block of the node-level network - like strength but giving extra weight to items connected to other well-connected items. "pca" First principal component of the member items' correlation matrix - a data-statistical weighting, blind to the estimated network. "factor" Standardized loadings of a one-factor model per cluster - the classical latent-variable weighting. The extraction method is chosen by fa_method and the correlation input respects cor_method (so polychoric factor analysis of ordinal items is one call). Clusters with fewer than 3 items (or non-converging fits) fall back to "pca" with a warning. "closeness", "betweenness" Within-cluster closeness / betweenness centrality of the item in the node-level network block (absolute weights). Betweenness can be all-zero in densely connected clusters; equal weights are used then, with a warning. "expected_influence" Mean <i>signed</i> connection to the other own-cluster members (Robinaugh et al. 2016) - like strength, but opposite-sign connections subtract, and negative expected influence marks reverse-keyed items.

"specificity" The misfit margin: own-cluster strength minus the strongest cross-cluster strength, floored at 0. Items that belong as much to another cluster contribute nothing to their composite - the weighting twin of the misfit diagnostic.

"item_total" Corrected item-total correlation: each item against the mean of the other (standardized) members - the classical scale-construction weighting.

Custom weightings: weighting also accepts a *named numeric vector* (one entry per item; absolute values are normalized within each cluster, signs flip items when signed = TRUE) or a *function* function(W_block, data_block, nodes) returning one numeric weight per item, evaluated per cluster.

Network-based and data-based weightings answer different questions; comparing them is informative - divergence means the network's view of the cluster differs from its latent-variable view. Schemes with inherently non-negative weights (equal, closeness, betweenness, specificity) keep the eigenvector-based item signs; sign-carrying schemes (eigen, pca, factor, expected_influence, item_total, custom) use their own.

cor_method	Character. Correlation type for estimation from raw data: "pearson" (default), "spearman", or "polychoric" (needs lavaan ; see Details).
signed	Logical. Flip reverse-keyed items in composites (default TRUE; see Details).
id_col	Character vector or NULL. Identifier column(s) to drop from data.frame input before analysis (e.g., the rid/actor columns produced by <code>convert_sequence_format</code> (format = "frequency"), whose output otherwise feeds this function directly as per-actor behavior profiles). Same convention as the association estimators.
fa_method	Character. Extraction method for weighting = "factor": "ml" (default) Maximum likelihood (stats::factanal on the cor_method-consistent correlation matrix). "paf" Iterated principal axis factoring (SMC start, communalities iterated on the reduced correlation matrix). "minres" Minimum residual / unweighted least squares (uniquenesses optimized to minimize squared off-diagonal residuals). "cfa" One-factor confirmatory model in lavaan (standardized loadings); with cor_method = "polychoric" the items are declared ordered, giving the categorical (DWLS) factor model. On well-behaved unidimensional clusters the four agree closely; divergence indicates Heywood-prone or non-unidimensional clusters.
...	Further arguments forwarded directly to the fa_method backend, exactly as that backend spells them (requires weighting = "factor"): lavaan arguments for "cfa" (estimator = "WLSMV", missing = "fiml", se = "robust", ...), stats::factanal() arguments for "ml", and max_iter / tol for "paf". Arguments managed internally (model, data, covmat, n.obs, factors) are ignored with a warning - the model is always the one-factor model per cluster, because the composite needs exactly one weight per item. In plot.mcml_pc(), print.mcml_pc() and summary.mcml_pc(): Additional arguments (ignored).
digits	In print.mcml_pc(): Number of digits to display (default 3). In plot.mcml_pc(): Number of digits for tile labels (default 2).

object For the `summary()` method: an object of class `mcml_pc`.

Details

Item diagnostics. Whenever raw data or a node-level network is available, every item's connection strength to *every* cluster is computed. The `item_loadings()` table reports, per item: its signed own-cluster loading, its composite weight, its strongest cross-cluster loading, and a misfit flag set when the cross-cluster loading exceeds the own-cluster loading - evidence the item is assigned to the wrong cluster. Misfit items trigger a warning; every aggregation silently inherits a bad membership, so fix the assignment rather than ignoring the flag.

Reverse-keyed items. With `signed = TRUE` (default), items whose summed within-cluster association is negative are flipped (their standardized values enter composites with weight sign -1), so a reverse-keyed item reinforces its cluster composite instead of cancelling it. Flips are reported in the `sign` column and via a warning.

Missing data. Composites are per-row weighted means over the *observed* members (weights renormalized per row); rows with no observed member yield NA and are dropped by the estimator with a message. Node-level estimation applies the estimators' own complete-case handling.

Ordinal items. `cor_method = "polychoric"` (requires the **lavaan** package) estimates the node-level and within-cluster networks from polychoric correlations - appropriate for Likert items. Composites are continuous sums, so the macro re-estimation uses Pearson correlations of the composites regardless.

Within-cluster networks follow within: "reestimate" (default) re-fits the estimator on the member columns alone - the honest conditional structure of the subsystem; "subnetwork" slices the node-level weight matrix and is descriptive (for `pcor/glasso` it retains conditioning on out-of-cluster nodes). Modes without raw data force "subnetwork". Singleton clusters get no within network (NULL).

Uncertainty. The composite/loadings macro network is a full `netobject` carrying its composite data, so `bootstrap_network(macro_network(fit))` (edge-weight CIs) and `vertex_bootstrap(macro_network(fit))` (network-level CIs) work directly; `vertex_compare(macro_network(fit1), macro_network(fit2))` compares two groups. `loading_stability` quantifies how stable the composite weights themselves are under case resampling.

All constituent networks are undirected (`meta$directed = FALSE`), so renderers that auto-detect directedness (e.g. `cograph::plot_mcml()`) draw the result without arrowheads.

Value

An object of class `"mcml_pc"` containing:

macro Cluster-level `netobject` ($k \times k$, undirected).

clusters Named list of within-cluster `netobjects` (NULL for singleton clusters).

cluster_members Named list of member node labels.

loadings Tidy item-diagnostic data frame (one row per node): `node`, `cluster`, `loading` (signed own-cluster), `weight`, `sign`, `max_cross`, `cross_cluster`, `misfit`. NULL only when no node-level network is available.

node_network The node-level `netobject` the aggregation was based on (kept for diagnostics and `loading_stability`).

data The raw data (data.frame) when available, else NULL.

meta List: aggregation (the name as the caller spelled it), method, within, weighting, fa_method, fa_args, scale, cor_method, signed, n_nodes, n_clusters, cluster_sizes, n_misfit, n_flipped, directed = FALSE, source = "pc", experimental = TRUE. method and weighting are NA on the descriptive paths that re-estimate nothing, and fa_method/fa_args are set only for weighting = "factor".

In `print.mcml_pc()`: x, invisibly.

In `summary.mcml_pc()`: Tidy data frame with one row per macro edge (upper triangle), columns from, to, weight.

In `plot.mcml_pc()`: A ggplot object.

Methods

- `plot.mcml_pc()`: Heatmap of the macro (cluster-level) weights with the package's diverging palette. For the two-layer network rendering use `cograph::plot_mcml()`, which accepts `mcml_pc` objects and draws them undirected.

References

- Escoufier, Y. (1973). Le traitement des variables vectorielles. *Biometrics*, 29(4), 751-760.
- Hotelling, H. (1936). Relations between two sets of variates. *Biometrika*, 28(3/4), 321-377.
- Robinaugh, D. J., Millner, A. J., & McNally, R. J. (2016). Identifying highly influential nodes in the complicated grief network. *Journal of Abnormal Psychology*, 125(6), 747-757.
- Christensen, A. P., & Golino, H. (2021). On the equivalency of factor and network loadings. *Behavior Research Methods*, 53, 1563-1580.
- Epskamp, S., & Fried, E. I. (2018). A tutorial on regularized partial correlation networks. *Psychological Methods*, 23(4), 617-634.

See Also

[build_mcml](#) for transition networks, [loading_stability](#) for composite-weight stability, [bootstrap_network](#) and [vertex_bootstrap](#) for uncertainty on the macro network.

Examples

```
set.seed(1)
n <- 200
sigma <- matrix(0.15, 6, 6)
sigma[1:3, 1:3] <- 0.5
sigma[4:6, 4:6] <- 0.5
diag(sigma) <- 1
z <- matrix(rnorm(n * 6), n, 6) %%% chol(sigma)
df <- as.data.frame(z)
names(df) <- c("a1", "a2", "a3", "b1", "b2", "b3")
clusters <- list(A = c("a1", "a2", "a3"), B = c("b1", "b2", "b3"))

fit <- build_mcml_pc(df, clusters, aggregation = "composite",
                    method = "cor")
```

```
fit          # macro (cluster-level) weights
summary(fit) # one row per macro edge
plot(fit)    # heatmap of the same weights
```

build_mlvar

Build a Multilevel Vector Autoregression (mlVAR) network

Description

Estimates three networks from ESM/EMA panel data, matching `mlVAR::mlVAR()` with `estimator = "lmer"`, `temporal = "fixed"`, `contemporaneous = "fixed"` at machine precision: (1) a directed temporal network of fixed-effect lagged regression coefficients, (2) an undirected contemporaneous network of partial correlations among residuals, and (3) an undirected between-subjects network of partial correlations derived from the person-mean fixed effects.

Usage

```
build_mlvar(
  data,
  vars,
  id,
  day = NULL,
  beep = NULL,
  lag = 1L,
  standardize = FALSE
)

## S3 method for class 'net_mlvar'
print(x, ...)

## S3 method for class 'net_mlvar'
summary(object, ...)
```

Arguments

<code>data</code>	A <code>data.frame</code> containing the panel data.
<code>vars</code>	Character vector of variable column names to model.
<code>id</code>	Character string naming the person-ID column.
<code>day</code>	Character string naming the day/session column, or <code>NULL</code> . When provided, lag pairs are only formed within the same day.
<code>beep</code>	Character string naming the measurement-occasion column, or <code>NULL</code> . When <code>NULL</code> , row position within each (<code>id</code> , <code>day</code>) is used.
<code>lag</code>	Integer. The lag order (default 1).

standardize	Logical. If TRUE, each variable is grand-mean centered and divided by its pooled SD <i>before</i> augmentation. Default FALSE, matching mlVAR::mlVAR(scale = FALSE) - the only setting for which numerical equivalence has been validated.
x	For the print() method: an object of class net_mlvar.
...	In print.net_mlvar() and summary.net_mlvar(): Unused; present for S3 consistency.
object	For the summary() method: an object of class net_mlvar.

Details

Estimation is delegated to `idiographic::fit_mlvar()` (the clean-room home of the temporal idiographic estimators), called with `estimator = "lmer"`, `temporal = "fixed"`, `contemporaneous = "fixed"`. The pipeline follows mlVAR's lmer path exactly:

1. Drop rows with NA in id/day/beep and optionally grand-mean standardize each variable.
2. Expand the per-(id, day) beep grid and right-join original values, producing the augmented panel (augData).
3. Add within-person lagged predictors (L1_*) and person-mean predictors (PM_*).
4. For each outcome variable fit `lmer(y ~ within + between-except-own-PM + (1 | id))` with `REML = FALSE`. Collect the fixed-effect temporal matrix B, between-effect matrix Gamma, random-intercept SDs (mu_SD), and lmer residual SDs.
5. Contemporaneous network: `cor2pcor(D %*% cov2cor(cor(resid)) %*% D)`.
6. Between-subjects network: `cor2pcor(pseudoinverse(forcePositive(D (I - Gamma))))`.

Validated to machine precision (`max_diff < 1e-10`) against mlVAR::mlVAR() on 25 real ESM datasets from openesm and 20 simulated configurations, and to exact equality (`max_diff == 0`) against the pre-delegation Nestimate implementation on all layers, coefficients, and observation counts across lag/standardize/day/beep configurations.

When the data carry no between-person variance (a random-intercept SD of zero), the between-subjects network is not estimable. Since 0.9.0 this raises a warning from **idiographic** and still returns the zero matrix by convention; before 0.9.0 the zero matrix was returned silently.

Value

A dual-class `c("net_mlvar", "netobject_group")` object - a named list of three full netobjects, one per network, plus model-level metadata stored as attributes. Each element is a standard `c("netobject", "cograph_network")` weight-matrix wrapper (no raw \$data), so `print()`, `summary()`, `coefs()`, and `cograph::splot(fit$temporal)` work directly. See **Dispatch limitation** for the verbs that do *not* work on this object (`plot()`, `bootstrap_network()`, `centrality()`, `reliability/stability`). Structure:

`fit$temporal` Directed netobject for the $d \times d$ matrix of fixed-effect lagged coefficients. `$weights[i, j]` is the effect of variable j at t -lag on variable i at t . `method = "mlvar_temporal"`, `directed = TRUE`.

`fit$contemporaneous` Undirected netobject for the $d \times d$ partial-correlation network of within-person lmer residuals. `method = "mlvar_contemporaneous"`, `directed = FALSE`.

`fit$between` Undirected netobject for the $d \times d$ partial-correlation network of person means, derived from $D(I - \Gamma)$. `method = "mlvar_between"`, `directed = FALSE`.

`attr(fit, "coefs") / coefs\(\)` Tidy data.frame with one row per (outcome, predictor) pair and columns outcome, predictor, beta, se, t, p, ci_lower, ci_upper, significant. Filter, sort, or plot with base R or the tidyverse. Retrieve with `coefs(fit)`.

`attr(fit, "n_obs")` Number of rows in the augmented panel after `na.omit`.

`attr(fit, "n_subjects")` Number of unique subjects remaining.

`attr(fit, "lag")` Lag order used.

`attr(fit, "standardize")` Logical; whether pre-augmentation standardization was applied.

In `print.net_mlvar()`: Invisibly returns `x`.

In `summary.net_mlvar()`: The tidy coefficient data.frame - the same table `coefs()` returns, with one row per (outcome, predictor) pair and columns outcome, predictor, beta, se, t, p, ci_lower, ci_upper, significant. Returned visibly, so calling `summary(fit)` at the console prints the matrices and then the table.

Dispatch limitation

There is no `plot()` method for `net_mlvar` - plot a single constituent (`cograph::splot(fit$temporal)`) instead. The three constituents are matrix-wrapped and carry no `$data`, so the data-resampling and data-reading verbs do **not** work on the fitted object or its parts: `bootstrap_network()`, `certainty()`, `network_reliability()`, `centrality_stability()` and `centrality()` all need the source panel. Extract a constituent and rebuild it through `build_network()` if you need those. Use `coefs()` for the tidy model output.

Methods

- `summary.net_mlvar()`: Prints the three weight matrices and the significant temporal edges, then returns the tidy coefficient table.

See Also

[build_network\(\)](#)

Examples

```
# A three-variable ESM panel: 20 people x 20 beeps. `tired` is driven by
# `happy` one beep earlier, so the temporal network should recover it.
if (requireNamespace("lme4", quietly = TRUE)) {
  set.seed(1)
  n_beep <- 20
  ar1 <- function(n, phi) as.numeric(stats::filter(stats::rnorm(n), phi,
                                                    method = "recursive"))
  panel <- do.call(rbind, lapply(seq_len(20), function(i) {
    happy <- ar1(n_beep, 0.4)
    data.frame(
      id    = i,
      beep  = seq_len(n_beep),
      happy = happy + stats::rnorm(1),
```

```

        calm = ar1(n_beep, 0.3) + stats::rnorm(1),
        tired = 0.5 * c(0, happy[-n_beep]) + stats::rnorm(n_beep) +
            stats::rnorm(1)
    )
  )))
fit <- build_mlvar(panel, vars = c("happy", "calm", "tired"),
  id = "id", beep = "beep")
fit
coefs(fit)
summary(fit)
}

```

build_mmm

Fit a Mixed Markov Model

Description

Discovers latent subgroups with different transition dynamics using Expectation-Maximization. Each mixture component has its own transition matrix. Sequences are probabilistically assigned to components.

Usage

```

build_mmm(
  data,
  k = 2L,
  n_starts = 50L,
  max_iter = 200L,
  tol = 1e-06,
  smooth = 0.01,
  seed = NULL,
  covariates = NULL,
  covariate_effect = c("em", "posthoc"),
  estimator = c("auto", "firth", "multinom", "chisq")
)

## S3 method for class 'net_mmm'
print(x, digits = 3L, ...)

## S3 method for class 'net_mmm'
summary(object, ...)

## S3 method for class 'net_mmm'
plot(x, type = c("posterior", "covariates"), combined = TRUE, ...)

## S3 method for class 'net_mmm_clustering'
print(x, digits = 3L, ...)

```

```
## S3 method for class 'net_mmm_clustering'
plot(
  x,
  type = c("posterior", "covariates", "predictors"),
  combined = TRUE,
  ...
)
```

Arguments

<code>data</code>	A data.frame (wide format), netobject, cograph_network, or tna model. For tna and cograph_network objects the stored (integer-encoded) data is extracted and decoded to state labels.
<code>k</code>	Integer. Whole finite number of mixture components, ≥ 2 . Default: 2.
<code>n_starts</code>	Integer. Positive whole finite number of random restarts. Default: 50.
<code>max_iter</code>	Integer. Positive whole finite maximum EM iterations per start. Default: 200.
<code>tol</code>	Numeric. Finite positive convergence tolerance. Default: 1e-6.
<code>smooth</code>	Numeric. Finite non-negative Laplace smoothing constant. Default: 0.01.
<code>seed</code>	Integer or NULL. Random seed.
<code>covariates</code>	Optional. Covariates integrated into the EM algorithm to model covariate-dependent mixing proportions. Accepts a string, character vector, formula, or data.frame (same forms as build_clusters). For netobject or cograph_network input, names are resolved against \$metadata first, so a typical call is <code>build_mmm(net, k = 3, covariates = "session_label")</code> . Unlike the post-hoc analysis in <code>build_clusters()</code> , these covariates directly influence cluster membership during EM estimation (see <code>covariate_effect</code>).
<code>covariate_effect</code>	How covariates enter the model. "em" (default) folds them into the EM as covariate-dependent mixing proportions, so they shape the cluster fit itself (and rows with missing covariates are dropped before fitting). "posthoc" fits a plain mixture on every sequence and uses the covariates only for the after-fit multinomial logit, so covariate values — and their missingness — never change which clusters are found. Ignored when covariates is NULL.
<code>estimator</code>	Multinomial fitter for the post-hoc covariate analysis (does not affect EM): "auto" (default) inspects the cluster x covariate cross-tab and falls back to "firth" only when any cell has fewer than 5 observations (separation risk), otherwise the much faster "multinom"; "firth" forces Firth's penalised likelihood via <code>brglm2::brmultinom</code> (finite under separation); "multinom" forces <code>nnet::multinom</code> (warns about separation risk); "chisq" runs descriptive tests (no logit). See build_clusters for full details.
<code>x</code>	For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_mmm</code> or <code>net_mmm_clustering</code> .
<code>digits</code>	In <code>print.net_mmm()</code> : Integer. Decimal places for floating-point statistics. Default 3. Non-breaking: <code>print(x)</code> keeps the same alignment as before. In <code>print.net_mmm_clustering()</code> : Integer. Decimal places for floating-point statistics. Default 3.

...	In <code>plot.net_mmm()</code> , <code>plot.net_mmm_clustering()</code> , <code>print.net_mmm()</code> , <code>print.net_mmm_clustering()</code> and <code>summary.net_mmm()</code> : Unsupported. Supplying unused arguments raises an error.
object	For the <code>summary()</code> method: an object of class <code>net_mmm</code> .
type	In <code>plot.net_mmm()</code> : Character. Plot type: "posterior" (default) or "covariates". In <code>plot.net_mmm_clustering()</code> : Character. One of "posterior" (default; histogram of max posterior probability per sequence, coloured by cluster), "covariates" or its alias "predictors" (covariate forest plot when <code>cluster_mmm()</code> was run with covariates).
combined	In <code>plot.net_mmm()</code> : Logical. For type = "covariates" only: when TRUE (default), covariate forest panels are combined into a single faceted plot; when FALSE, a list of separate ggplots is returned. In <code>plot.net_mmm_clustering()</code> : Logical. For type in "covariates" or "predictors" only: when TRUE (default), forest panels are combined into a single faceted plot; when FALSE, a list of separate ggplots is returned.

Value

An object of class `net_mmm` with components:

data The full N-row sequence frame used for estimation.

models List of `netobjects`, one per component. Each component carries the rows assigned to that component in its `$data` slot, while its transition matrix is the EM-estimated component transition matrix.

k Number of components.

mixing Numeric vector of mixing proportions.

posterior N x k matrix of posterior probabilities.

assignments Integer vector of hard assignments (1..k).

quality List: `avepp` (per-class), `avepp_overall`, `entropy`, `relative_entropy`, `classification_error`, `class_entropy`.

log_likelihood, BIC, AIC, ICL Model fit statistics.

n_params Number of free parameters behind BIC/AIC/ICL. With `covariate_effect = "em"` it grows by $(k - 1) * p$ for p covariate columns.

iterations, converged EM iterations used by the retained fit and whether it met `tol`.

states Character vector of state names.

n_sequences Number of sequences actually fitted (rows with missing covariates are dropped under `covariate_effect = "em"`).

covariates The post-hoc covariate analysis (list), or NULL when `covariates = NULL`.

network_method, build_args, httna_partition Provenance kept from `netobject` / HTNA input so per-cluster networks can be rebuilt the same way; NULL otherwise.

metadata The `netobject`'s per-sequence metadata, one row per fitted sequence in the row order of posterior, so `session_ids` can name each sequence; NULL for other input.

In `print.net_mmm()` and `print.net_mmm_clustering()`: The input object, invisibly.

In `summary.net_mmm()`: A per-component summary data.frame. The class and visibility depend on whether the model was fitted with covariates:

No covariates A plain data.frame with one row per component and columns component, prior, n_assigned, mean_posterior, avepp, returned *visibly* (so it auto-prints after the printed summary block).

With covariates A tidy_covariates/data.frame (the tidied covariate table, with the per-component stats attached), returned *invisibly*.

In both cases the printed summary (model fit, per-cluster transition matrices, optional covariate profiles) is emitted as a side effect.

In `plot.net_mmm()`: A ggplot object, invisibly; for type = "covariates" with combined = FALSE, a list of ggplot objects named by cluster (invisibly).

In `plot.net_mmm_clustering()`: A ggplot object, invisibly; for type = "covariates" / "predictors" with combined = FALSE, a list of ggplot objects named by cluster (invisibly).

Initial states

The first sequence column has special status: it is read directly as the per-sequence initial state (`init_state[i] <- match(raw_data[i, state_cols[1L]], states)`). The function does **not** scan forward to the first non-missing position, and it does not apply any na_syms-style symbol conversion (unlike `build_clusters`). The state vocabulary is built from the unique non-NA values across all columns, so if your data uses a sentinel character such as "*" or "%" for missing cells, that sentinel becomes a real state and the first column reads it as a valid initial state. If you want padded leading missings to be treated as missing, recode them to NA before calling `build_mmm()` (then `match()` returns NA, which the EM treats as an uninformative initial distribution), or left-trim the leading missings so each sequence's first column carries an observed state.

Methods

- `plot.net_mmm_clustering()`: Plot routines for the MMM clustering metadata attached to the `netobject_group` that `build_network` materializes from a `cluster_mmm` fit (or that `cluster_network` returns directly with `cluster_by = "mmm"`). Mirrors the type-driven surface of `plot.net_clustering` but covers only the metrics the EM fit produces – there is no distance matrix on an MMM clustering, so "silhouette" / "mds" / "heatmap" aren't defined here and the dispatcher raises a clear error if you ask for one of those on an MMM result.
- `print.net_mmm()`: Compact summary of a Mixed Markov Model fit. Header carries dimensions and information criteria; cluster table carries N, mixing share, and per-cluster average posterior probability (AvePP). Layout matches `print.net_clustering` so distance- and model-based clusterings can be compared at a glance.
- `print.net_mmm_clustering()`: Prints the clustering metadata attached to the `netobject_group` that `build_network` materializes from a `cluster_mmm` fit (`attr(grp, "clustering")`). Layout mirrors `print.net_clustering`: a one-line dimension header, a quality line with AvePP / entropy / classification error, information criteria, and a per-cluster table.

See Also

`compare_mmm`, `build_network`

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                   V2 = sample(c("A", "B", "C"), 30, TRUE))
mmm <- build_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)
mmm

seqs <- data.frame(
  V1 = sample(LETTERS[1:3], 30, TRUE), V2 = sample(LETTERS[1:3], 30, TRUE),
  V3 = sample(LETTERS[1:3], 30, TRUE), V4 = sample(LETTERS[1:3], 30, TRUE)
)
mmm <- build_mmm(seqs, k = 2, seed = 42)
print(mmm)
summary(mmm)
```

build_mogen

*Build Multi-Order Generative Model (MOGen)***Description**

Constructs higher-order De Bruijn graphs from sequential trajectory data and selects the optimal Markov order using AIC, BIC, or likelihood ratio tests.

Usage

```
build_mogen(
  data,
  max_order = 5L,
  criterion = c("aic", "bic", "lrt"),
  lrt_alpha = 0.01
)

## S3 method for class 'net_mogen'
print(x, ...)

## S3 method for class 'net_mogen'
summary(object, ...)

## S3 method for class 'net_mogen'
plot(x, type = c("ic", "likelihood"), ...)
```

Arguments

data	A data.frame (rows = trajectories, columns = time points), a list of character/numeric vectors (one per trajectory), a tna object, or a netobject with sequence data. For tna/netobject, numeric state IDs are automatically converted to label names.
------	--

max_order	Integer. Maximum Markov order to test (default 5). Must be a whole number; a non-integer value (e.g. 2.7) is an error rather than being silently truncated. A max_order at or above the longest trajectory is capped at (longest path - 1) with a message.
criterion	Character. Model selection criterion: "aic" (default), "bic", or "lrt" (likelihood ratio test).
lrt_alpha	Numeric. Significance threshold for LRT (default 0.01).
x	For the print() and plot() methods: an object of class net_mogen.
...	In plot.net_mogen(): Additional arguments passed to plot . In print.net_mogen() and summary.net_mogen(): Additional arguments (ignored).
object	For the summary() method: an object of class net_mogen.
type	Character. Plot type: "ic" (default) or "likelihood".

Details

At order k , nodes are k -tuples of states and edges represent transitions between overlapping k -tuples. The model tests increasingly complex Markov orders and selects the one that best balances fit and parsimony.

Value

An object of class `c("net_mogen", "cograph_network")` with components:

optimal_order Selected optimal Markov order.

criterion Which criterion was used for selection.

orders Integer vector of tested orders (0 to max_order, after any capping).

aic Named numeric vector of AIC values per order.

bic Named numeric vector of BIC values per order.

log_likelihood Named numeric vector of log-likelihoods.

dof Named integer vector of cumulative DOF per model.

layer_dof Named integer vector of per-layer DOF.

transition_matrices List of row-stochastic transition matrices (index 1 = order 0, held as the marginal named numeric vector).

count_matrices List of the matching raw count matrices, same indexing; read by [mogen_transitions\(\)](#).

states Unique first-order states.

n_paths Number of trajectories.

n_observations Total number of state observations.

weights `cograph_network` weight matrix: the transition matrix of the selected optimal order (a 1 x n matrix named "marginal" when the optimal order is 0). Its dimnames are the internal k -gram keys (states joined by a non-printing separator), not arrow notation.

nodes `data.frame` (id, label, name) of the optimal-order De Bruijn nodes.

edges `cograph_network` edge `data.frame` with integer from/to node indices and a numeric weight. The readable arrow-notation table is [mogen_transitions\(\)](#).

directed Logical. Always TRUE.

n_nodes, n_edges Counts for the optimal-order graph.

meta cograph_network metadata list.

node_groups Always NULL.

In `print.net_mogen()` and `plot.net_mogen()`: The input object, invisibly.

In `summary.net_mogen()`: A per-order model-selection data.frame with columns `order`, `layer_dof`, `cum_dof`, `loglik`, `aic`, `bic`, `best` ("AIC"/"BIC"/"AIC+BIC" marker) and `selected` ("<--" on the chosen order), returned visibly; the summary text is printed as a side effect.

References

Scholtes, I. (2017). When is a Network a Network? Multi-Order Graphical Model Selection in Pathways and Temporal Networks. *KDD 2017*.

Gote, C., Casiraghi, G., Schweitzer, F., & Scholtes, I. (2023). Predicting variable-length paths in networked systems using multi-order generative models. *Applied Network Science*, 8, 68.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
mg <- build_mogen(seqs, max_order = 2)
```

```
trajs <- list(c("A","B","C","D"), c("A","B","D","C"),
             c("B","C","D","A"), c("C","D","A","B"))
m <- build_mogen(trajs, max_order = 3)
print(m)
plot(m)
```

build_network

Build a Network

Description

Universal network estimation function that supports both transition networks (relative, frequency, co-occurrence) and association networks (correlation, partial correlation, graphical lasso). Uses the global estimator registry, so custom estimators can also be used.

Usage

```
build_network(
  data,
  method,
  actor = NULL,
  action = NULL,
```

```

time = NULL,
session = NULL,
order = NULL,
codes = NULL,
group = NULL,
format = "auto",
window_size = 3L,
mode = c("non-overlapping", "overlapping"),
scaling = NULL,
threshold = 0,
level = NULL,
time_threshold = 900,
timezone = "UTC",
predictability = TRUE,
state_cols = NULL,
metadata_cols = NULL,
start = FALSE,
end = FALSE,
params = list(),
labels = NULL,
...
)

## S3 method for class 'netobject'
print(x, ...)

## S3 method for class 'netobject_group'
print(x, digits = 3L, ...)

## S3 method for class 'netobject_ml'
print(x, ...)

## S3 method for class 'netobject'
summary(object, ...)

## S3 method for class 'netobject_group'
summary(object, combined = TRUE, ...)

## S3 method for class 'summary.netobject'
print(x, ...)

## S3 method for class 'summary.netobject_group'
print(x, ...)

```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance). A fitted <code>net_clustering</code> or <code>net_mmm</code> object is also accepted: the per-cluster networks are (re)built and a <code>netobject_group</code>
------	---

is returned.

method	Character. Required, except when data is a <code>net_clustering</code> or <code>net_mmm</code> object, where the fitted object's own network method is used. Name of a registered estimator. Built-in methods: "relative", "frequency", "co_occurrence", "cor", "pcor", "glasso", "ising", "mgm", "attention", "wtna", "wtna_cooccurrence", "ngram", "gap", "reverse". The last three mirror <code>tna::build_model()</code> types "n-gram" (adjacent pairs counted once per n-gram window containing them; <code>params = list(n_gram = 2)</code>), "gap" (pairs up to <code>max_gap + 1</code> positions apart, weighted by <code>1 / distance</code> ; <code>params = list(max_gap = 1)</code>) and "reverse" (reply network: the transpose of "frequency"; <code>params = list(weighted = FALSE)</code>). All three return raw weights; add <code>scaling = "normalize"</code> for row probabilities. Aliases: "tna" and "transition" map to "relative"; "ftna" and "counts" map to "frequency"; "cna" and "wcna" map to "co_occurrence"; "corr" and "correlation" map to "cor"; "partial" maps to "pcor"; "ebicglasso" and "regularized" map to "glasso"; "isingfit" maps to "ising"; "atna" maps to "attention"; "mixed" and "mixed_graphical" map to "mgm"; "wtna_transition" maps to "wtna"; "co-occurrence" maps to "co_occurrence"; "n-gram" and "n_gram" map to "ngram".
actor	Character. Name of the actor/person ID column for sequence grouping. Default: NULL.
action	Character. Name of the action/state column (long format). Default: NULL.
time	Character. Name of the time column (long format). Default: NULL.
session	Character. Name of the session column. Default: NULL.
order	Character. Name of the ordering column. Default: NULL.
codes	Character vector. Column names of one-hot encoded states (for onehot format). Default: NULL.
group	Character. Name of a grouping column for per-group networks. Returns a <code>netobject_group</code> (named list of <code>netobjects</code>). Default: NULL.
format	Character. Input format: "auto", "wide", "long", or "onehot". Default: "auto".
window_size	Integer. Window size for one-hot windowing. Default: 3L.
mode	Character. Windowing mode for one-hot input only: "non-overlapping" or "overlapping". Has no effect on wide or long sequence data (only the one-hot/wtna path reads it). Default: "non-overlapping".
scaling	Character vector or NULL. Post-estimation scaling to apply (in order). Options: "minmax", "max", "rank", "normalize". Can combine: <code>c("rank", "minmax")</code> . Default: NULL (no scaling).
threshold	Numeric. Absolute values below this are set to zero in the result matrix. Default: 0 (no thresholding).
level	Character or NULL. Multilevel decomposition for the undirected association methods (cor, pcor, glasso); a directed estimator errors. One of NULL, "between", "within", "both". Requires an id column, supplied either as actor or as <code>params\$id / params\$id_col</code> . Default: NULL.

time_threshold	Numeric or FALSE. Maximum time gap (seconds) for long format session splitting. Set to FALSE to switch session-interval splitting off, so each actor (or actor-session) forms a single sequence. Default: 900.
timezone	Character. Olson time zone used to interpret naive timestamps in long-format data (offset-bearing timestamps such as ...Z or +02:00 are converted from their offset). Passed to prepare . Default: "UTC".
predictability	Logical. If TRUE (default), compute and store node predictability (R-squared) for undirected association methods (glasso, pcor, cor). Stored in \$predictability and auto-displayed as donuts by <code>cograph::splot()</code> .
state_cols	Character vector or NULL. Explicit names of columns to classify as state columns in the returned netobject's \$data slot. When provided, all other columns of the cleaned input go to \$metadata. Auto-detection (values-in-nodes heuristic) is bypassed. Use this when a metadata column happens to contain values that overlap with node names (e.g. condition labels "A", "B", "C" and nodes "A", "B", "C") and auto-detection would misclassify it. Default: NULL (auto-detect).
metadata_cols	Character vector or NULL. Explicit names of columns to force into the \$metadata slot. The remaining columns are auto-detected as state via the values-in-nodes rule. Cannot overlap with state_cols. Default: NULL.
start	Boundary marker prepended to every sequence as an explicit start state (a pure source: no incoming edges, every sequence's first transition is start -> first_observed). FALSE (default) adds nothing; TRUE uses the label "Start"; a single string uses that string as the label. Only valid for the transition methods (relative, frequency, co_occurrence, attention, ngram, gap, reverse); errors otherwise (wtna included).
end	Boundary marker placed in the single cell after each sequence's last observed (non-NA) state, as an explicit terminal state (a pure sink: no outgoing edges, no self-loop – distinct from mark_terminal_state , which fills all trailing NAs into an absorbing state). FALSE (default) adds nothing; TRUE uses the label "End"; a single string uses that string as the label. Same method restriction as start.
params	Named list. Method-specific parameters passed to the estimator function (e.g. <code>list(gamma = 0.5)</code> for glasso, or <code>list(format = "wide")</code> for transition methods). This is the key composability feature: downstream functions like bootstrap or grid search can store and replay the full params list without knowing method internals. Transition estimators accept tna-style sequence options such as weighted and concat (and the low-level begin_state / end_state, of which start / end are the public form – see those arguments). Column-like entries in params (action, id, id_col, actor, time, session, order, cols, codes, and group) are resolved before format detection and must name existing columns. If the same column role is supplied both directly and through params, the names must agree.
labels	Optional name -> label remap applied after construction. Accepts a 2-column data.frame (name, label), a named character vector <code>c(name = "label")</code> , or a named list. Rewrites \$nodes\$label and dimnames(weights). Unmapped names pass through unchanged.

...	Additional arguments passed to the estimator function. In <code>print.netobject()</code> , <code>print.netobject_group()</code> and <code>print.netobject_ml()</code> : Additional arguments (ignored). In <code>print.summary.netobject()</code> , <code>print.summary.netobject_group()</code> , <code>summary.netobject()</code> and <code>summary.netobject_group()</code> : Ignored.
x	For the <code>print()</code> method: an object of class <code>netobject</code> , <code>netobject_group</code> or <code>netobject_ml</code> (or its <code>summary()</code>).
digits	Integer. Decimal places for the weight summary. Default 3. Non-breaking: <code>print(x)</code> keeps the same shape as before, with the addition of a weight-range column.
object	For the <code>summary()</code> method: an object of class <code>netobject</code> or <code>netobject_group</code> .
combined	Logical. Combine into one wide data.frame? Default TRUE.

Details

The function works as follows:

1. Resolves method aliases to canonical names.
2. Validates explicit column arguments before any format guessing.
3. Retrieves the estimator function from the global registry.
4. For association methods with `level` specified, decomposes the data (between-person means or within-person centering).
5. Calls the estimator: `do.call(fn, c(list(data = data), params))`.
6. Applies scaling and thresholding to the result matrix.
7. Extracts edges and constructs the `netobject`.

For long-format transition data, supplying `action` without `actor` is allowed and treats all rows as one sequence in row/time order. The function warns because a one-sequence transition network is not recommended and cannot be validated by bootstrap or other confirmatory tests.

Value

An object of class `c("netobject", "cograph_network")` containing:

data The state columns of the cleaned input data, as a data frame.

metadata Data frame of the non-state columns of the cleaned input (and, for long input, the per-sequence metadata), or NULL.

weights The estimated network weight matrix.

nodes Data frame with columns `id`, `label`, `name`, `x`, `y`. Node labels are in `$nodes$label`.

edges Data frame of non-zero edges with integer `from/to` (node IDs) and numeric `weight`.

directed Logical. Whether the network is directed.

method The resolved method name.

params The params list used (for reproducibility).

scaling The scaling applied (or NULL).

threshold The threshold applied.

n_nodes Number of nodes.
n_edges Number of non-zero edges.
level Decomposition level used (or NULL).
build_args The resolved column/format arguments (actor, action, time, session, order, codes, format, window_size, mode) used for this build.
meta List with source, layout, and tna metadata (cograph-compatible).
node_groups Node groupings data frame, or NULL.
predictability Named numeric vector of R-squared predictability values per node (for undirected association methods when predictability = TRUE). NULL for directed methods.

Method-specific extras (e.g. precision_matrix, cor_matrix, frequency_matrix, initial, lambda_selected, etc.) are preserved from the estimator output.

When level = "both", returns an object of class "netobject_ml" with \$between and \$within sub-networks and a \$method field. level = "between" or "within" returns a single netobject estimated on the decomposed data.

When group is supplied (or data is a net_clustering / net_mmm object), returns an object of class "netobject_group": a named list of netobjects, one per group, carrying the grouping column in attr(x, "group_col").

In print.netobject(), print.netobject_group() and print.netobject_ml(): The input object, invisibly.

In summary.netobject(): A data.frame with columns metric and value, of class c("summary.netobject", "data.frame").

In summary.netobject_group(): Either a data.frame (one column per group) or a named list of summary.netobject objects, of class c("summary.netobject_group", ...).

In print.summary.netobject(): x, invisibly.

In print.summary.netobject_group(): x, invisibly.

Methods

- print.netobject_group(): Compact summary of a netobject_group. Header surfaces the source (a clustering attached by [cluster_network](#) or [cluster_mmm](#), or a plain split by group_col). The per-group table carries node and edge counts, weight range, and – when a clustering attribute is present – N and percentage of sequences per cluster (matching the layout used by [print.net_clustering](#) and [print.net_mmm](#)).
- summary.netobject(): Computes node count, edge count, density, mean shortest-path distance, mean and SD of in/out strength, mean and SD of in/out degree, in/out degree centralization (Freeman), and reciprocity. Mirrors the metric set returned by tna::summary.tna() so a Nestimate netobject and the equivalent tna model report numerically identical descriptive metrics.
- summary.netobject_group(): Returns one summary per constituent network. With combined = TRUE (default) the per-group tables are joined into a single wide data.frame with one column per group; with combined = FALSE returns a named list.

See Also

[register_estimator](#), [list_estimators](#), [bootstrap_network](#)

Examples

```

seqs <- data.frame(V1 = c("A","B","C","A"), V2 = c("B","C","A","B"))
net <- build_network(seqs, method = "relative")
net

# Transition network (relative probabilities)
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
print(net)

# Association network (glasso)
freq_data <- convert_sequence_format(seqs, format = "frequency")
net_glasso <- build_network(freq_data, method = "glasso",
                           params = list(gamma = 0.5, nlambda = 50))

# With scaling
net_scaled <- build_network(seqs, method = "relative",
                           scaling = c("rank", "minmax"))

```

build_pcor

*Build a Partial Correlation Network***Description**

Convenience wrapper for `build_network(method = "pcor")`. Computes partial correlations from numeric data.

Usage

```
build_pcor(data, ...)
```

Arguments

<code>data</code>	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance). A fitted <code>net_clustering</code> or <code>net_mmm</code> object is also accepted: the per-cluster networks are (re)built and a <code>netobject_group</code> is returned.
<code>...</code>	Additional arguments passed to build_network .

Value

A `netobject` (see [build_network](#)).

See Also[build_network](#)**Examples**

```
data(srl_strategies)
net <- build_pcor(srl_strategies)
```

build_simplicial	<i>Build a Simplicial Complex</i>
------------------	-----------------------------------

Description

Constructs a simplicial complex from a network or higher-order pathway object. Two construction methods are available:

- **Clique complex** ("clique"): every clique in the thresholded non-zero graph becomes a simplex. Edges with absolute weight $\geq$ threshold are retained. The standard bridge from graph theory to algebraic topology.
- **Pathway complex** ("pathway"): each higher-order pathway from a net_hon or net_hypa becomes a simplex.

For type = "vr" (or alias "rips"), the input is treated as a non-negative distance / dissimilarity matrix and a Vietoris-Rips filtration is constructed: each k-simplex σ enters at $\max_{(i,j) \in \sigma} d(i,j)$. Use max_scale to cap the filtration diameter; edges with $d(i,j) > \text{max_scale}$ are excluded. Filtration values are attached as \$filtration on the returned object so persistent_homology() can read them directly.

Usage

```
build_simplicial(
  x,
  type = "clique",
  threshold = 0,
  max_dim = 10L,
  max_pathways = NULL,
  anomaly = c("all", "over", "under"),
  max_scale = NULL,
  ...
)

## S3 method for class 'simplicial_complex'
print(x, ...)

## S3 method for class 'simplicial_complex'
plot(x, combined = TRUE, ...)
```

Arguments

<code>x</code>	A square matrix, <code>tna</code> , <code>netobject</code> , <code>net_hon</code> , <code>net_hypa</code> , or <code>net_mogen</code> . For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>simplicial_complex</code> .
<code>type</code>	Construction type: "clique" (default), "pathway", or "vr" (alias "rips").
<code>threshold</code>	For type = "clique": minimum non-zero absolute edge weight to include an edge (default 0). Edges below this are ignored; zero-weight non-edges are never included. Ignored for type = "vr" (use <code>max_scale</code> instead) and for type = "pathway".
<code>max_dim</code>	Maximum simplex dimension (default 10). Must be a single non-negative integer. A k-simplex has k+1 nodes.
<code>max_pathways</code>	For type = "pathway": maximum number of pathways to include, ranked by count (HON) or ratio (HYPA). NULL includes all. Default NULL.
<code>anomaly</code>	For HYPA pathway complexes, which anomaly direction to include: "all" (default), "over", or "under". Under-represented HYPA paths are ranked by smallest observed/expected ratio; over-represented paths are ranked by largest ratio.
<code>max_scale</code>	For type = "vr": maximum edge length to include in the filtration. NULL (default) uses <code>max(d)</code> .
<code>...</code>	Additional arguments passed to <code>build_hon()</code> when <code>x</code> is a <code>tna/netobject</code> with type = "pathway". In <code>plot.simplicial_complex()</code> : Ignored. In <code>print.simplicial_complex()</code> : Additional arguments (unused).
<code>combined</code>	When TRUE (default), the four panels are stitched into a 2x2 <code>gtable</code> via <code>gridExtra::arrangeGrob</code> and drawn. When FALSE, returns a named list of the four <code>ggplots</code> (<code>f_vector</code> , <code>betti</code> , <code>degree</code> , <code>degree_heatmap</code>) so each can be printed, saved, or re-laid-out independently.

Value

A `simplicial_complex` object - a list with:

simplices List of integer vectors, one per simplex, each holding the (sorted) node indices it spans.

Every face of every simplex is present, including all 0-simplices (isolated vertices included).

nodes Character vector of node labels; simplices index into it.

n_nodes, n_simplices Integer counts.

dimension Integer. Highest simplex dimension present (a k-simplex has k+1 nodes).

f_vector Named integer vector `dim_0`, `dim_1`, ... - the number of simplices of each dimension.

density Numeric. `n_simplices` divided by the number of simplices a complete complex on `n_nodes` would have up to `dimension`.

mean_dim Numeric. Mean simplex dimension.

type "clique", "pathway", or "vr".

For type = "vr" two further elements are attached: `$filtration` (numeric, parallel to `$simplices`: the value at which each simplex enters) and `$max_scale` (the cap actually used). `persistent_homology()` consumes them directly.

In `print.simplicial_complex()`: The input object, invisibly.

In `plot.simplicial_complex()`: A grid grob (invisibly) when `combined = TRUE`; a named list of four ggplots when `combined = FALSE`.

Methods

- `plot.simplicial_complex()`: Produces a four-panel summary: f-vector, Betti numbers, simplicial degree ranking, and degree-by-dimension heatmap.

See Also

[betti_numbers](#), [persistent_homology](#), [simplicial_degree](#), [q_analysis](#)

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
print(sc)
betti_numbers(sc)

# Vietoris-Rips on a distance matrix:
d <- 1 - mat
diag(d) <- 0
sc_vr <- build_simplicial(d, type = "vr", max_scale = 0.6)
```

build_tna

Build a Transition Network (TNA)

Description

Convenience wrapper for `build_network(method = "relative")`. Computes row-normalized transition probabilities from sequence data.

Usage

```
build_tna(data, start = FALSE, end = FALSE, ...)
```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance). A fitted <code>net_clustering</code> or <code>net_mmm</code> object is also accepted: the per-cluster networks are (re)built and a <code>netobject_group</code> is returned.
------	--

start	Boundary marker prepended to every sequence as an explicit start state (a pure source: no incoming edges, every sequence's first transition is start -> first_observed). FALSE (default) adds nothing; TRUE uses the label "Start"; a single string uses that string as the label. Only valid for the transition methods (relative, frequency, co_occurrence, attention, ngram, gap, reverse); errors otherwise (wtna included).
end	Boundary marker placed in the single cell after each sequence's last observed (non-NA) state, as an explicit terminal state (a pure sink: no outgoing edges, no self-loop – distinct from mark_terminal_state , which fills all trailing NAs into an absorbing state). FALSE (default) adds nothing; TRUE uses the label "End"; a single string uses that string as the label. Same method restriction as start.
...	Additional arguments passed to build_network .

Value

A netobject (see [build_network](#)).

See Also

[build_network](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
net <- build_tna(seqs)
```

casedrop_reliability *Edge-weight Case-dropping Stability*

Description

Computes a **CS-coefficient for the edge-weight vector** of a network: the maximum proportion of cases (rows of `x$data`) that can be dropped while the flattened edge-weight vector of the re-estimated network still correlates with the original above threshold in at least certainty of iterations.

Usage

```
casedrop_reliability(
  x,
  iter = 1000L,
  drop_prop = seq(0.1, 0.9, by = 0.1),
  threshold = 0.7,
  certainty = 0.95,
  method = c("spearman", "pearson", "kendall"),
  include_diag = FALSE,
  seed = NULL
```

```

)

## S3 method for class 'net_casedrop_reliability'
print(x, digits = 3, ...)

## S3 method for class 'net_casedrop_reliability'
summary(object, ...)

## S3 method for class 'net_casedrop_reliability_group'
print(x, ...)

## S3 method for class 'net_casedrop_reliability_group'
summary(object, drop_prop = NULL, ...)

## S3 method for class 'summary.net_casedrop_reliability_group'
print(x, ...)

## S3 method for class 'net_casedrop_reliability'
plot(x, combined = TRUE, ...)

## S3 method for class 'net_casedrop_reliability_group'
plot(
  x,
  metric = c("correlation", "mean_abs_dev", "median_abs_dev", "max_abs_dev"),
  ...
)

```

Arguments

<code>x</code>	A <code>netobject</code> , <code>cograph_network</code> , <code>netobject_group</code> , or <code>mcml</code> . For the group types this function iterates over each element and returns a named list. For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_casedrop_reliability</code> or <code>net_casedrop_reliability_group</code> (or its <code>summary()</code>).
<code>iter</code>	Integer. Iterations per drop proportion. Default 1000.
<code>drop_prop</code>	Numeric vector of proportions to evaluate. Each entry must lie strictly between 0 and 1. Default <code>seq(0.1, 0.9, by = 0.1)</code> . In <code>summary.net_casedrop_reliability_group()</code> : Drop proportion at which to report the four metrics (mean +/- sd per network). Must be one of the drop proportions the object was built with. Defaults to the object's median grid value (the stored grid is used, not an assumed 0.7); pass an explicit value not in the grid to get an error listing the available proportions.
<code>threshold</code>	Numeric in $[0, 1]$. Minimum edge-vector correlation for an iteration to count as stable. Default 0.7.
<code>certainty</code>	Numeric in $[0, 1]$. Required fraction of iterations whose correlation must exceed threshold for a drop proportion to qualify. Default 0.95.
<code>method</code>	Correlation method: "pearson" (weight magnitudes), "spearman" (ranks, robust to scale), or "kendall". Default "spearman" because edge weights often span several orders of magnitude and rank stability is the typical target.

include_diag	Logical. Include diagonal (self-loop) edges in the edge vector. Default FALSE.
seed	Optional integer for reproducibility.
digits	Digits to display. Default 3.
...	In <code>plot.net_casedrop_reliability()</code> , <code>plot.net_casedrop_reliability_group()</code> , <code>print.net_casedrop_reliability()</code> , <code>print.summary.net_casedrop_reliability_group()</code> , <code>summary.net_casedrop_reliability()</code> and <code>summary.net_casedrop_reliability_group()</code> : Additional arguments (ignored). In <code>print.net_casedrop_reliability_group()</code> : Ignored.
object	For the <code>summary()</code> method: an object of class <code>net_casedrop_reliability</code> or <code>net_casedrop_reliability_group</code> .
combined	When TRUE (default), all four metrics are shown in one ggplot via <code>facet_wrap(~metric)</code> . When FALSE, returns a named list of four single-panel ggplots, one per metric.
metric	Which metric to plot. One of "correlation" (default), "mean_abs_dev", "median_abs_dev", "max_abs_dev".

Details

Complements `centrality_stability()`: that function asks whether centrality *rankings* are stable; this one asks whether the *edge-weight structure itself* is stable. For MCML-derived networks where each row of `$data` is one transition, this is case-dropping of **edges**.

For each drop_prop `p` and each iteration, a size `n_cases * (1 - p)` subset of `$data` rows is selected **without replacement**, the network is re-estimated using the same method/scaling/threshold as the input, and the upper/lower-triangle (directed: all off-diagonal entries) of the new weight matrix is flattened and correlated with the corresponding vector of the original matrix. The correlation method defaults to Spearman for robustness to the wide dynamic range of transition probabilities.

Unlike bootstrap CIs, case-dropping does not estimate sampling variance and so does not rely on the i.i.d. assumption. This makes it the appropriate robustness check for **edgelist-derived** networks (where rows of `$data` lack actor grouping), since dropping rows at random is a well-posed operation regardless of within-actor correlation.

Value

An object of class `net_casedrop_reliability` with:

`cs` Scalar CS-coefficient - the maximum drop proportion for which the edge-vector correlation remains $\geq$ threshold in at least certainty of iterations. Zero if no proportion qualifies.

`summary` Tidy data frame, one row per metric by drop proportion, with columns `metric` ("mean_abs_dev", "median_abs_dev", "correlation", "max_abs_dev"), `drop_prop`, `mean`, `sd`, `median`, `mad`, `q025`, `q975`.

`metrics` Named list of four `iter x length(drop_prop)` matrices, one per metric, holding the raw per-iteration values.

`correlations` `iter x length(drop_prop)` matrix of per- iteration correlations (the correlation entry of metrics).

`drop_prop`, `threshold`, `certainty`, `iter`, `method`, `include_diag` Inputs.

`n_cases` Number of cases resampled from (sequences for transition methods, rows of `$data` otherwise).

`n_edges` Length of the edge vector assessed.

A `netobject_group` or `mcml` input instead returns a `net_casedrop_reliability_group`: a named list of one result per constituent network.

When the original edge vector has zero variance a warning is issued and the object is returned with `cs = 0`, an empty summary, and all-NA metric matrices.

In `print.net_casedrop_reliability()`: The input `x` invisibly.

In `summary.net_casedrop_reliability()`: A tidy data frame with columns `metric`, `drop_prop`, `mean`, `sd` summarising edge-weight stability across case-dropping iterations.

In `summary.net_casedrop_reliability_group()`: A data frame with one row per network containing `cor`, `mean_abs_dev`, `median_abs_dev`, `max_abs_dev` formatted as "mean +/- sd".

In `plot.net_casedrop_reliability()`: A ggplot object, or a named list of four ggplots when `combined = FALSE`.

In `plot.net_casedrop_reliability_group()`: A ggplot object.

Methods

- `plot.net_casedrop_reliability()`: Plots the four model-level reliability metrics across drop proportions: `correlation`, `mean_abs_dev`, `median_abs_dev`, `max_abs_dev`. Each panel shows the per-iteration mean with a ribbon at mean +/- sd. The correlation panel includes a dashed horizontal line at the user's threshold (default 0.7).
- `plot.net_casedrop_reliability_group()`: Overlay of per-cluster correlation curves across drop proportions. One colour per sub-network; ribbons show mean +/- sd across iterations. Dashed horizontal line marks the stability threshold (default 0.7).

References

Epskamp, S., Borsboom, D., & Fried, E. I. (2018). Estimating psychological networks and their accuracy: A tutorial paper. *Behavior Research Methods* 50(1), 195-212. doi:10.3758/s13428017-08621

See Also

[centrality_stability\(\)](#), [bootstrap_network\(\)](#).

Examples

```
set.seed(1)
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE),
  V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
es <- casedrop_reliability(net, iter = 50, drop_prop = c(0.1, 0.3, 0.5),
  seed = 1)
```

```
print(es)
```

centrality_stability *Centrality Stability Coefficient (CS-coefficient)*

Description

Estimates the stability of centrality indices under case-dropping. For each drop proportion, sequences are randomly removed and the network is re-estimated. The correlation between the original and subset centrality values is computed. The CS-coefficient is the maximum proportion of cases that can be dropped while maintaining a correlation above threshold in at least certainty of bootstrap samples.

For transition methods, uses pre-computed per-sequence count matrices for fast resampling. Strength centralities (InStrength, OutStrength) are computed directly from the matrix without igraph.

Usage

```
centrality_stability(
  x,
  measures = c("InStrength", "OutStrength", "Betweenness"),
  iter = 1000L,
  drop_prop = seq(0.1, 0.9, by = 0.1),
  threshold = 0.7,
  certainty = 0.95,
  method = "pearson",
  centrality_fn = NULL,
  loops = FALSE,
  normalize = FALSE,
  invert = TRUE,
  normalize_diffusion = TRUE,
  seed = NULL
)

## S3 method for class 'net_stability'
print(x, ...)

## S3 method for class 'net_stability_group'
print(x, ...)

## S3 method for class 'net_stability_group'
summary(object, ...)

## S3 method for class 'net_stability'
summary(object, ...)

## S3 method for class 'net_stability'
plot(x, ...)
```

Arguments

<code>x</code>	A netobject from <code>build_network</code> , a <code>cograph_network</code> , or a <code>netobject_group</code> / <code>mcml</code> (each constituent network is assessed and a <code>net_stability_group</code> is returned). For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_stability</code> or <code>net_stability_group</code> .
<code>measures</code>	Character vector. Centrality measures to assess. Defaults to <code>c("InStrength", "OutStrength", "Betweenness")</code> . Pass "all" for every built-in measure: "OutStrength", "InStrength", "ClosenessIn", "ClosenessOut", "Closeness", "Betweenness", "BetweennessRSP", "Diffusion", and "Clustering". The legacy aliases "InCloseness" and "OutCloseness" are also accepted. Custom measures beyond these are valid only when a <code>centrality_fn</code> is supplied to resolve them.
<code>iter</code>	Integer. Number of bootstrap iterations per drop proportion (default: 1000).
<code>drop_prop</code>	Numeric vector. Proportions of cases to drop (default: <code>seq(0.1, 0.9, by = 0.1)</code>).
<code>threshold</code>	Numeric. Minimum correlation to consider stable (default: 0.7).
<code>certainty</code>	Numeric. Required proportion of iterations above threshold (default: 0.95).
<code>method</code>	Character. Correlation method: "pearson", "spearman", or "kendall" (default: "pearson").
<code>centrality_fn</code>	Optional function. A custom centrality function that takes a weight matrix and returns a named list of centrality vectors. When NULL (default), all built-in measures are computed internally: "InStrength"/"OutStrength" via <code>colSums/rowSums</code> , "Betweenness"/"ClosenessIn"/"ClosenessOut"/"Closeness" via an internal Floyd-Warshall shortest-path routine, and "BetweennessRSP", "Diffusion" and "Clustering" from the weight matrix directly. When provided, the function is called as <code>centrality_fn(mat)</code> and is used only for requested measures that are not one of the built-ins; it should return a named list (e.g., <code>list(my_metric = ...)</code>).
<code>loops</code>	Logical. If FALSE (default), self-loops (diagonal) are excluded from centrality computation. This does not modify the stored matrix.
<code>normalize</code>	Logical. Range-normalize all requested measures using the same transformation as <code>tna::centralities(normalize = TRUE)</code> . Default: FALSE.
<code>invert</code>	Logical. Invert weights for shortest-path measures? Default: TRUE, matching <code>tna</code> .
<code>normalize_diffusion</code>	Logical. Range-normalize Diffusion even when <code>normalize = FALSE</code> . Default: TRUE.
<code>seed</code>	Integer or NULL. RNG seed for reproducibility.
<code>...</code>	In <code>plot.net_stability()</code> , <code>print.net_stability()</code> , <code>print.net_stability_group()</code> , <code>summary.net_stability()</code> and <code>summary.net_stability_group()</code> : Additional arguments (ignored).
<code>object</code>	For the <code>summary()</code> method: an object of class <code>net_stability_group</code> or <code>net_stability</code> .

Value

An object of class "net_stability": a list with

cs Named numeric vector of CS-coefficients, one per retained measure.

correlations Named list of iter x length(drop_prop) matrices of correlation values, one per retained measure.

measures Character vector of the measures actually assessed (see the zero-variance rule below).

drop_prop Drop proportions used.

threshold Stability threshold.

certainty Required certainty level.

iter Number of iterations.

method Correlation method.

A netobject_group or mcml input instead returns a "net_stability_group": a named list of one net_stability per constituent network.

Zero-variance measures are handled by two different rules, both long-standing behaviour. When *some* requested measures have zero variance on the original network (for example "OutStrength" on a row-normalised transition network), those measures are **dropped**: \$cs, \$correlations and \$measures cover only the retained ones. When *every* requested measure has zero variance a warning is issued and **all** requested names are returned with cs = 0 and all-NA correlation matrices.

In print.net_stability(): The input object, invisibly.

In print.net_stability_group(): The input x invisibly.

In summary.net_stability_group(): A data frame with columns group, measure, drop_prop, mean_cor, sd_cor, prop_above.

In summary.net_stability(): A data frame with columns measure, drop_prop, mean_cor, sd_cor, prop_above.

In plot.net_stability(): A ggplot object (invisibly).

Methods

- plot.net_stability(): Plots mean correlation vs drop proportion for each centrality measure. The CS-coefficient is marked where the curve crosses the threshold.
- summary.net_stability(): Returns the mean correlation at each drop proportion for each measure.
- summary.net_stability_group(): Per-network stability as a tidy data frame. Stacks summary() results for each network with a group column.

References

Epskamp, S., Borsboom, D., & Fried, E. I. (2018). Estimating psychological networks and their accuracy: A tutorial paper. *Behavior Research Methods* 50(1), 195-212. doi:10.3758/s13428017-08621

See Also

[build_network](#), [network_reliability](#)

Examples

```
seqs <- data.frame(
  T1 = c("plan", "code", "debug", "plan", "test", "code"),
  T2 = c("code", "debug", "code", "plan", "code", "test"),
  T3 = c("debug", "code", "plan", "code", "debug", "plan"),
  T4 = c("test", "plan", "test", "debug", "plan", "code")
)
net <- build_network(seqs, method = "relative")
cs <- centrality_stability(net, iter = 10, drop_prop = 0.3, seed = 1)

set.seed(1)
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
cs <- centrality_stability(net, iter = 100, seed = 42,
  measures = c("InStrength", "OutStrength"))
print(cs)
```

certainty

Analytic certainty of network edges (Bayesian Dirichlet-Multinomial)

Description

Closed-form alternative to [bootstrap_network](#) for transition networks. Models the outgoing transitions from each state as a Dirichlet-Multinomial process: with a Jeffreys prior the posterior for state i is $\text{Dirichlet}(c_i + \text{prior})$, so each edge is marginally Beta and its posterior mean, standard deviation, credible interval and stability decision are available analytically. No resampling, so it runs in microseconds.

The return value has the same structure as [bootstrap_network](#) (same slots and summary columns) and carries class `c("net_certainty", "net_bootstrap")`, so `summary()` and any code that consumes a `net_bootstrap` object work unchanged.

Usage

```
certainty(
  x,
  prior = 0.5,
  ci_level = 0.05,
  inference = c("stability", "threshold"),
  consistency_range = c(0.75, 1.25),
```

```

    edge_threshold = NULL
)

## S3 method for class 'net_certainty'
print(x, ...)
```

Arguments

x	A netobject from build_network using a transition-probability method ("relative" / "tna"), or a netobject_group. For the print() method: an object of class net_certainty.
prior	Numeric. Dirichlet prior concentration added to every cell (default 0.5, the Jeffreys prior).
ci_level	Numeric in (0,1). Tail level for credible intervals and the stability decision (default 0.05, i.e. a 95% interval). Named to match bootstrap_network().
inference	Character. "stability" (default) tests whether the posterior keeps the edge within a multiplicative consistency_range of its weight; "threshold" tests whether the edge exceeds edge_threshold.
consistency_range	Numeric vector of length 2. Multiplicative bounds for stability inference (default c(0.75, 1.25)).
edge_threshold	Numeric or NULL. Fixed threshold for inference = "threshold". If NULL, defaults to the 10th percentile of non-zero edge weights.
...	In print.net_certainty(): Additional arguments (ignored).

Details

Certainty (this function), stability ([bootstrap_network](#)) and reliability ([reliability](#)) answer different questions about an edge: how precisely it is pinned down by the observed counts, whether it survives resampling the sequences, and whether it is consistent across split-halves. Certainty and stability agree on homogeneous data; certainty is over-confident when the data are a mixture of latent classes, because it treats transitions clustered within a sequence as independent.

Value

For a netobject: an object of class c("net_certainty", "net_bootstrap") with the same fields as [bootstrap_network](#): original, mean, sd, p_values, significant, ci_lower, ci_upper, cr_lower, cr_upper, summary, model, method, params, ci_level, inference, consistency_range, edge_threshold, plus prior, ci_method = "analytic" and iter = NA (no iterations).

For a netobject_group: a named list of those objects, one per constituent network, of class c("net_certainty_group", "net_bootstrap_group", "list").

In print.net_certainty(): The input object, invisibly.

References

Johnston, L. & Jendoubi, T. (2026). How Delivery Mode Reshapes Resource Engagement: A Bayesian Differential Network Analysis. TNA Workshop 2026.

See Also

[bootstrap_network](#), [bayes_compare](#), [network_reliability](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A", "C", "B"), V2 = c("B", "C", "B", "A", "C"),
                  V3 = c("C", "A", "C", "B", "A"))
net <- build_network(seqs, method = "relative")
cert <- certainty(net)
cert
summary(cert)
```

chain_structure

Qualitative structure of a discrete-time Markov chain

Description

Computes properties that depend only on the transition matrix support, not on any starting distribution: state classification, communicating classes, periods, irreducibility / aperiodicity / regularity / reversibility, hitting probabilities, and absorption analysis when absorbing states exist.

Usage

```
chain_structure(x, normalize = TRUE, tol = 1e-10)

## S3 method for class 'chain_structure'
print(x, ...)

## S3 method for class 'chain_structure'
plot(x, show_values = TRUE, digits = 2L, ...)

## S3 method for class 'chain_structure'
summary(object, ...)

## S3 method for class 'chain_structure_group'
print(x, ...)

## S3 method for class 'chain_structure_group'
summary(object, ...)

## S3 method for class 'summary_chain_structure'
print(x, ...)
```

Arguments

x	A netobject, cograph_network, tna model, transition matrix, or sequence data.frame (passed through build_network() with method = "relative"). A netobject_group is also accepted and analysed constituent by constituent. For the print() and plot() methods: an object of class chain_structure, chain_structure_group or summary_chain_structure.
normalize	Logical. If TRUE (default), rows of the transition matrix are renormalized to sum to 1 before analysis (see passage_time() for the same convention).
tol	Numerical tolerance for the reversibility check (detailed balance) and for treating near-zero entries as zero when building the support graph (which drives classification, communicating_classes, period, and hitting_probabilities). It does not govern the absorbing-state test: a state is absorbing only when $P[i, i]$ equals 1 to an internal fixed tolerance of $.Machine$double.eps^{0.5}$, independent of tol (so raising tol to ignore tiny transition probabilities never reclassifies a near-deterministic state as absorbing). Default $1e-10$.
...	In plot.chain_structure(), print.chain_structure(), summary.chain_structure() and summary.chain_structure_group(): Ignored. In print.chain_structure_group(): Forwarded to print.chain_structure. In print.summary_chain_structure(): Forwarded to print.data.frame.
show_values	Logical. If TRUE (default), prints the numeric probability inside each cell. Set FALSE for large state spaces ($n > 10$) where labels overlap.
digits	Integer. Decimal places for in-cell labels.
object	For the summary() method: an object of class chain_structure or chain_structure_group.

Details

Built specifically as a diagnostic to run *before* trusting the output of [passage_time\(\)](#) or [markov_stability\(\)](#). Both implicitly assume a regular chain (irreducible + aperiodic) so that the stationary distribution is unique and meaningful. Use is_regular to check.

The fundamental-matrix absorption math follows Kemeny & Snell (1976); the hitting-probability linear system follows Norris (1997).

Value

For a netobject_group, a c("chain_structure_group", "list"): a named list holding one chain_structure per constituent network, with its own print() and summary() methods.

Otherwise a chain_structure object: a list with elements

states Character vector of state names.

classification Named character vector. One of "absorbing", "recurrent", "transient" per state.

communicating_classes List of state-name vectors. Each sublist is a strongly connected component of the support graph.

recurrent_classes Subset of communicating_classes that are closed (no transitions leaving the class).

`transient_classes` Subset that are not closed.

`absorbing_states` Character vector of states with $P[i, i] = 1$ (tested exactly, to within `.Machine$double.eps^0.5`; the user-facing `tol` does not relax this).

`period` Named integer vector. Period of each recurrent state; NA for transient states.

`is_irreducible` Logical. TRUE iff there is exactly one communicating class.

`is_aperiodic` Logical. TRUE iff every recurrent state has period 1.

`is_regular` Logical. `is_irreducible && is_aperiodic`.

`is_reversible` Logical or NA. TRUE iff the chain satisfies detailed balance against its stationary distribution. NA for non-irreducible chains (no unique stationary).

`hitting_probabilities` $n \times n$ matrix. $[i, j] = P(\text{ever reach } j \text{ starting from } i)$, computed over the same `tol`-thresholded support graph that drives classification so the two are mutually consistent (a state classified "absorbing"/closed never shows hitting probability to states outside its class).

`absorption_probabilities` $n_{\text{transient}} \times n_{\text{absorbing}}$ matrix or NULL if no transient $\rightarrow$ absorbing pathway exists. $[i, j] = P(\text{eventual absorption in } j \mid \text{start in } i)$.

`mean_absorption_time` Named numeric vector or NULL. Expected number of steps until absorption from each transient state.

`P` The (possibly normalized) transition matrix used.

In `print.chain_structure()`, `print.chain_structure_group()` and `print.summary_chain_structure()`: `x` invisibly.

In `plot.chain_structure()`: A ggplot object.

In `summary.chain_structure()`: A data.frame with one row per state, of class `c("summary_chain_structure", "data.frame")`, carrying the chain-level flags (`is_regular`, `is_irreducible`, `is_aperiodic`, `is_reversible`, `n_classes`, `absorbing_states`) as attributes, which its `print()` method shows as a header. Columns as described above.

In `summary.chain_structure_group()`: A data.frame with columns `group`, `state`, `classification`, `period`, `persistence`, `return_probability`, `sojourn_steps`, plus `stationary_probability` if all groups are irreducible and `mean_absorption_time` if any group has absorbing states.

Methods

- `plot.chain_structure()`: Renders the hitting-probability matrix as a heatmap, with rows and columns ordered by communicating class so the block structure is visible at a glance. State labels along both axes are coloured by classification (absorbing / recurrent / transient). The subtitle summarises the chain-level properties (regular, reversible).
- `print.chain_structure()`: Prints a compact chain-level header. For the full per-state table, call `summary()` on the same object.
- `print.chain_structure_group()`: One header line per group, followed by each group's per-state table (via `summary.chain_structure`).
- `print.summary_chain_structure()`: Prints a one-line chain header followed by the tidy per-state table.

- `summary.chain_structure()`: Returns a single data.frame with one row per state, combining every per-state metric `chain_structure()` computes. Always includes state, classification, period, persistence (the diagonal of the transition matrix), return_probability (the diagonal of the hitting matrix) and sojourn_steps ($1 / (1 - \text{persistence})$), which is Inf for an absorbing state). Adds the chain's stationary_probability when the chain is irreducible, and absorption columns when it has any absorbing states: absorption_probability for a single absorbing state or one absorbed_in_<state> column per state when there are several, plus mean_absorption_time.
- `summary.chain_structure_group()`: Produces a single tidy data.frame with one row per (group, state) combination, combining classification, persistence, sojourn, and – when applicable – stationary or mean-absorption-time columns. Useful for side-by-side reporting of `chain_structure()` across the members of a `netobject_group`.

Plot colours

Cell colour encodes $P(\text{ever reach } j \mid \text{start at } i)$. The diagonal uses the return-time convention ($P(\text{return to } j \text{ in } \geq 1 \text{ steps})$), matching `markovchain::hittingProbabilities`. A non-irreducible chain shows zero off-block entries – visual evidence of one-way doors between behavioural phases. An absorbing chain shows a column of 1's for the absorbing state.

Summary columns

Columns are ordered for readability: identifiers first, classification second, dynamic per-state metrics last.

References

- Kemeny, J. G. and Snell, J. L. (1976). *Finite Markov Chains*. Springer-Verlag.
- Norris, J. R. (1997). *Markov Chains*. Cambridge University Press.

See Also

[passage_time\(\)](#), [markov_stability\(\)](#), [build_network\(\)](#)

Examples

```
net <- build_network(as.data.frame(trjectories), method = "relative")
cs <- chain_structure(net)
print(cs)

summary(cs)
```

chatgpt_srl	<i>ChatGPT Self-Regulated Learning Scale Scores</i>
-------------	---

Description

Scale scores on five self-regulated learning (SRL) constructs for 1,000 responses generated by ChatGPT to a validated SRL questionnaire. Part of a larger dataset comparing LLM-generated responses to human norms across seven large language models.

Usage

chatgpt_srl

Format

- A data frame with 1,000 rows and 5 columns:
- CSU** Numeric. Comprehension and Study Understanding scale mean.
- IV** Numeric. Intrinsic Value scale mean.
- SE** Numeric. Self-Efficacy scale mean.
- SR** Numeric. Self-Regulation scale mean.
- TA** Numeric. Task Avoidance scale mean.

Source

Vogelsmeier, L.V.D.E., Oliveira, E., Misiejuk, K., Lopez-Pernas, S., & Saqr, M. (2025). Delving into the psychology of Machines: Exploring the structure of self-regulated learning via LLM-generated survey responses. *Computers in Human Behavior*, 173, 108769. doi:10.1016/j.chb.2025.108769

Examples

```
net <- build_network(chatgpt_srl, method = "glasso",
                     params = list(gamma = 0.5))
net
```

clique_expansion	<i>Clique expansion of a hypergraph</i>
------------------	---

Description

Projects a [net_hypergraph](#) to a standard pairwise [netobject](#) (the *clique expansion* - also called the "downgrade" of a hypergraph to a dyadic graph). Each hyperedge of size k contributes 1 (or its weight) to every pair of its members. The resulting edge weight $W[i, j]$ equals the number of hyperedges containing both i and j (binary incidence) or the sum of incidence products (weighted incidence).

Usage

```
clique_expansion(hg, weighted = TRUE)
```

Arguments

hg	A net_hypergraph object as returned by build_hypergraph() or bipartite_groups() .
weighted	Logical. If TRUE (default), use the hypergraph's incidence values directly (so weighted hypergraphs from bipartite_groups() produce weighted projections). If FALSE, binarise the incidence first so $W[i, j]$ is just the count of shared hyperedges.

Details

The clique expansion is the standard "loss-y but lossless-on-pairwise" projection: it preserves *which pairs co-occurred* and *how often* but discards the higher-order grouping. Comparing `clique_expansion(hg)` to a directly-estimated pairwise network (e.g. via [cooccurrence\(\)](#) on the same data) quantifies how much information was carried by the hyperedge structure.

Computed in one BLAS call via `tcrossprod(incidence)`; runs in $O(n_{\text{nodes}}^2 * n_{\text{hyperedges}})$ time, fast for typical sizes.

Closes the I/O cycle: event data -> [bipartite_groups\(\)](#) -> `clique_expansion()` -> any function that accepts a [netobject](#) (centrality, bootstrap, clustering, plotting via [cograph](#)).

Value

A [netobject](#) (also [cograph_network](#)) with `method = "clique_expansion"`, undirected, with weighted symmetric adjacency $W = \text{incidence} \%*\% t(\text{incidence})$ and zero diagonal. The standard [netobject](#) fields are present (`$weights`, `$nodes`, `$edges` - one row per non-zero upper-triangle cell with integer from/to node indices and weight - `$n_nodes`, `$n_edges`, `$meta`); `$params` records source, `weighted`, `n_hyperedges` and `hypergraph_size_distribution`.

Note

(experimental) Validated against `tcrossprod(incidence)` with zero diagonal. No external R package exposes clique expansion as a primitive; the implementation is a direct one-line restatement of the definition.

References

Tian, H., & Zafarani, R. (2024). Higher-order networks representation and learning: A survey. *ACM SIGKDD Explorations Newsletter* 26(1), 1-18.

See Also

[build_hypergraph\(\)](#), [bipartite_groups\(\)](#), [build_network\(\)](#).

Examples

```
df <- data.frame(
  player = c("A", "B", "C", "A", "B", "D", "C", "D", "E"),
  session = c("S1", "S1", "S1", "S2", "S2", "S3", "S3", "S3", "S3")
)
hg <- bipartite_groups(df, player = "player", group = "session")
net <- clique_expansion(hg)
extract_edges(net, threshold = 1)
```

cluster_choice

Cluster Choice – sweep k, dissimilarity and method

Description

One-call sweep across any combination of k, dissimilarity metric, and clustering algorithm for distance-based sequence clustering. Mirrors [compare_mmm](#) for model-based clustering: returns a data frame with one row per swept configuration, a best marker on the silhouette-max row in the print method, and a `plot()` that adapts to the swept axes.

Usage

```
cluster_choice(
  data,
  k = 2:5,
  dissimilarity = "hamming",
  method = "ward.D2",
  ...
)

## S3 method for class 'cluster_choice'
print(x, digits = 3L, ...)

## S3 method for class 'cluster_choice'
summary(object, ...)

## S3 method for class 'cluster_choice'
plot(
```

```

x,
type = c("auto", "lines", "bars", "heatmap", "tradeoff", "facet"),
abbrev = FALSE,
combined = TRUE,
...
)

```

Arguments

<code>data</code>	Sequence data (data frame or matrix) – forwarded to build_clusters .
<code>k</code>	Integer vector of cluster counts to sweep. Default 2:5. Each value must be ≥ 2 and $\leq n - 1$.
<code>dissimilarity</code>	Character vector of dissimilarity metrics. Use "all" to expand to every supported metric: <code>c("hamming", "osa", "lv", "dl", "lcs", "qgram", "cosine", "jaccard", "jw")</code> . Default "hamming".
<code>method</code>	Character vector of clustering algorithms. Use "all" to expand to every supported method: <code>c("pam", "ward.D2", "ward.D", "complete", "average", "single", "mcquitty", "median", "centroid")</code> . Default "ward.D2".
<code>...</code>	Other arguments forwarded to build_clusters (<code>weighted</code> , <code>lambda</code> , <code>q</code> , <code>p</code> , <code>seed</code> , <code>na_syms</code> , <code>covariates</code> , <code>estimator</code>). Note: <code>weighted = TRUE</code> only works with <code>dissimilarity = "hamming"</code> and is rejected up-front when sweeping mixed dissimilarities. In <code>plot.cluster_choice()</code> , <code>print.cluster_choice()</code> and <code>summary.cluster_choice()</code> : Unsupported. Supplying unused arguments raises an error.
<code>x</code>	For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>cluster_choice</code> .
<code>digits</code>	Integer. Decimal places for floating-point columns. Default 3L.
<code>object</code>	For the <code>summary()</code> method: an object of class <code>cluster_choice</code> .
<code>type</code>	Character. One of "auto" (default), "lines", "bars", "heatmap", "tradeoff", "facet".
<code>abbrev</code>	Logical. If TRUE, dissimilarity and method names shown on tick labels and point labels are shortened (e.g. "hamming" -> "ham", "ward.D2" -> "wD2"). The legend shows the full canonical name. Default FALSE.
<code>combined</code>	Only meaningful for <code>type = "facet"</code> . When TRUE (default), all methods are shown in one ggplot via <code>facet_wrap(~ method)</code> . When FALSE, returns a named list of single-panel ggplots, one per method.

Value

A `cluster_choice` object (a `data.frame` subclass) with one row per (`k`, `dissimilarity`, `method`) combination and columns:

`k`, `dissimilarity`, `method` The configuration for that row.

`silhouette` Overall average silhouette width (from `cluster::silhouette`, computed inside [build_clusters](#)).

`mean_within_dist` Size-weighted mean of within-cluster distances, in the units of the row's dissimilarity.

min_size, max_size, size_ratio Cluster-size balance bounds and their ratio (max / min).

In `print.cluster_choice()`: The input object, invisibly.

In `summary.cluster_choice()`: A data frame with the swept configurations, all metrics, and a best character column flagging the silhouette-max row.

In `plot.cluster_choice()`: A ggplot object, invisibly; for `type = "facet"` with `combined = FALSE`, a named list of ggplots.

Methods

- `plot.cluster_choice()`: Six explicit chart types plus a smart "auto" default. The user picks the shape; the function does not editorialise (no "best" annotation, no interpretive subtitles, no inferred recommendation).

Plot types

Type cheat-sheet:

"auto" Default. Picks one of the others based on which axes were swept. k-only -> "lines"; one categorical axis swept -> "bars"; k plus one categorical -> "lines"; k plus two categoricals -> "facet"; both categoricals without k -> "heatmap".

"lines" Silhouette across k (and `mean_within_dist` when k is the only swept axis), one line per non-k axis when present.

"bars" Horizontal bar chart of silhouette per axis level. Bars sorted by silhouette.

"heatmap" Tiled silhouette across two categorical axes. Requires both dissimilarity and method swept.

"tradeoff" Scatter: silhouette (y) vs `size_ratio` (x). Works for any sweep; labels each point.

"facet" Lines vs k, colour by one categorical axis, facet by another. Requires k plus two categoricals.

Asking for a type the data can't support raises an error pointing at the alternatives.

See Also

[build_clusters](#), [compare_mmm](#) for the model-based equivalent, [cluster_diagnostics](#) for the post-fit diagnostic surface on a single clustering.

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 40, TRUE),
                  V2 = sample(c("A", "B", "C"), 40, TRUE))
cluster_choice(seqs, k = 2:4)

# Sweep dissimilarities at fixed k
cluster_choice(seqs, k = 3, dissimilarity = c("hamming", "lcs", "jaccard"))

# Full grid of k x dissimilarity
cluster_choice(seqs, k = 2:4, dissimilarity = c("hamming", "lcs"))
```

```
# "all" sentinel
cluster_choice(seqs, k = 3, dissimilarity = "all")
```

cluster_diagnostics *Cluster Diagnostics*

Description

Unified entry point for clustering quality information. Returns a `net_cluster_diagnostics` object that normalises the diagnostic surface across distance-based and model-based clusterings – you no longer have to know which fields live on `net_clustering` vs. `net_mmm` vs. the slim `net_mmm_clustering` attribute of a `netobject_group`.

Usage

```
cluster_diagnostics(x, ...)

## S3 method for class 'net_cluster_diagnostics'
print(x, digits = 3L, ...)

## S3 method for class 'net_cluster_diagnostics'
plot(x, type = NULL, ...)

## S3 method for class 'net_cluster_diagnostics'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

<code>x</code>	A <code>net_clustering</code> , <code>net_mmm</code> , <code>netobject_group</code> (with <code>attr(, "clustering")</code> attached by <code>cluster_network()</code> or <code>build_network(net_mmm)</code>), or <code>net_mmm_clustering</code> . For the <code>print()</code> , <code>plot()</code> and <code>as.data.frame()</code> methods: an object of class <code>net_cluster_diagnostics</code> .
<code>...</code>	Unsupported. Supplying unused arguments raises an error. In <code>as.data.frame.net_cluster_diagnostics()</code> and <code>print.net_cluster_diagnostics()</code> : Unsupported. Supplying unused arguments raises an error. In <code>plot.net_cluster_diagnostics()</code> : Forwarded to the underlying plot method.
<code>digits</code>	Integer. Decimal places for floating-point statistics. Default 3L.
<code>type</code>	Character. Forwarded to the underlying plot method. Valid values for distance: "silhouette" (default), "mds", "heatmap", "predictors". Valid values for mmm: "posterior" (default), "covariates" / "predictors".
<code>row.names, optional</code>	Standard <code>as.data.frame</code> arguments (ignored).

Details

The returned object carries:

family Either "distance" or "mmm".

k, n, sizes Number of clusters, number of sequences, sizes vector.

per_cluster A data.frame – one row per cluster, columns differ by family. Distance: cluster, size, pct, mean_within_dist, sil_mean. MMM: cluster, size, pct, mix_pct, avepp, class_err_pct.

overall A named list of family-specific summary metrics (silhouette for distance; avepp_overall, entropy, classification_error for MMM).

ics For MMM: a list with BIC, AIC, ICL. NULL for distance.

metadata Method / dissimilarity / weighted / lambda etc.

source The original clustering object, kept by reference so plot() can delegate without recomputing anything.

Value

cluster_diagnostics() returns a net_cluster_diagnostics object: a list carrying family, k, n, sizes, the per_cluster data frame (one row per cluster), overall, ics, metadata and source, as detailed above. as.data.frame() on that object returns the per_cluster data frame itself – one row per cluster, with family-specific columns.

In print.net_cluster_diagnostics(): The input object, invisibly.

In plot.net_cluster_diagnostics(): Whatever the underlying plot method returns: a ggplot object, invisibly; or, for the covariate forest views called with combined = FALSE, a list of ggplot objects named by cluster (invisibly).

Methods

- plot.net_cluster_diagnostics(): Delegates to the original clustering object's plot method (plot.net_clustering for distance-based diagnostics, plot.net_mmm_clustering or plot.net_mmm for model-based). The diagnostics object itself stores no plot geometry – it just keeps a reference to the source so the existing visual layer is reused.
- print.net_cluster_diagnostics(): Prints a uniform header, family-specific quality / IC line, and a per-cluster table. Layout matches print.net_clustering and print.net_mmm.

See Also

print.net_cluster_diagnostics, plot.net_cluster_diagnostics, compare_mmm for k-sweep model selection (MMM only).

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                  V2 = sample(c("A", "B", "C"), 30, TRUE))
cl <- build_clusters(seqs, k = 2, method = "ward.D2")
cluster_diagnostics(cl)
```

```
fit <- cluster_mmm(seqs, k = 2, n_starts = 1, max_iter = 20, seed = 1)
cluster_diagnostics(fit)
as.data.frame(cluster_diagnostics(fit))
```

cluster_mmm

Cluster sequences using Mixed Markov Models

Description

Fits a mixture of Markov chains to sequence data and returns the fitted `net_mmm` clustering object. The fit retains assignments, posterior probabilities, mixing proportions, information criteria, and the fitted component models.

Usage

```
cluster_mmm(
  data,
  k = 2L,
  n_starts = 50L,
  max_iter = 200L,
  tol = 1e-06,
  smooth = 0.01,
  seed = NULL,
  covariates = NULL,
  covariate_effect = c("em", "posthoc"),
  estimator = c("auto", "firth", "multinom", "chisq"),
  cluster_by = "mmm",
  ...
)
```

Arguments

<code>data</code>	A <code>data.frame</code> (wide format), <code>netobject</code> , <code>cograph_network</code> , or <code>tna</code> model. For <code>tna</code> and <code>cograph_network</code> objects the stored (integer-encoded) data is extracted and decoded to state labels.
<code>k</code>	Integer. Whole finite number of mixture components, ≥ 2 . Default: 2.
<code>n_starts</code>	Integer. Positive whole finite number of random restarts. Default: 50.
<code>max_iter</code>	Integer. Positive whole finite maximum EM iterations per start. Default: 200.
<code>tol</code>	Numeric. Finite positive convergence tolerance. Default: $1e-6$.
<code>smooth</code>	Numeric. Finite non-negative Laplace smoothing constant. Default: 0.01.
<code>seed</code>	Integer or <code>NULL</code> . Random seed.

covariates	Optional. Covariates integrated into the EM algorithm to model covariate-dependent mixing proportions. Accepts a string, character vector, formula, or data.frame (same forms as build_clusters). For netobject or cograph_network input, names are resolved against \$metadata first, so a typical call is build_mmm(net, k = 3, covariates = "session_label"). Unlike the post-hoc analysis in build_clusters(), these covariates directly influence cluster membership during EM estimation (see covariate_effect).
covariate_effect	How covariates enter the model. "em" (default) folds them into the EM as covariate-dependent mixing proportions, so they shape the cluster fit itself (and rows with missing covariates are dropped before fitting). "posthoc" fits a plain mixture on every sequence and uses the covariates only for the after-fit multinomial logit, so covariate values — and their missingness — never change which clusters are found. Ignored when covariates is NULL.
estimator	Multinomial fitter for the post-hoc covariate analysis (does not affect EM): "auto" (default) inspects the cluster x covariate cross-tab and falls back to "firth" only when any cell has fewer than 5 observations (separation risk), otherwise the much faster "multinom"; "firth" forces Firth's penalised likelihood via brglm2::brmultinom (finite under separation); "multinom" forces nnet::multinom (warns about separation risk); "chisq" runs descriptive tests (no logit). See build_clusters for full details.
cluster_by	Character. Accepted only as "mmm" (the default). Present so cluster_mmm() and cluster_network() share the same call shape; any other value raises an error pointing at cluster_network .
...	Unsupported. Supplying unused arguments raises an error.

Details

To materialize one network per fitted cluster, pass the result to [build_network](#) or use cluster_network(..., cluster_by = "mmm") for fitting and network construction in one call.

Value

A fitted net_mmm clustering object. This is the same object contract returned by [build_mmm](#). For HTNA input, its preserved actor partition is restored when the fit is materialized with [build_network](#) or Nestimate::as_htna().

See Also

[build_mmm](#), [build_network](#), and [cluster_network](#) for fitting and immediately materializing per-cluster networks

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                  V2 = sample(c("A", "B", "C"), 30, TRUE))
fit <- cluster_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)
fit
cluster_diagnostics(fit)
```

```
# Visualise with sequence_plot
seqs <- data.frame(
  V1 = sample(LETTERS[1:3], 40, TRUE),
  V2 = sample(LETTERS[1:3], 40, TRUE),
  V3 = sample(LETTERS[1:3], 40, TRUE)
)
fit <- cluster_mmm(seqs, k = 2)
sequence_plot(fit, type = "index")
```

cluster_network	<i>Cluster data and build per-cluster networks in one step</i>
-----------------	--

Description

Combines sequence clustering and network estimation into a single call. Clusters the data using the specified algorithm, then calls [build_network](#) on each cluster subset.

Usage

```
cluster_network(data, k, cluster_by = "pam", dissimilarity = "hamming", ...)
```

Arguments

data	Sequence data. Accepts a data frame, matrix, or netobject. See build_clusters for supported formats.
k	Integer. Number of clusters.
cluster_by	Character. Clustering algorithm passed to build_clusters 's method parameter ("pam", "ward.D2", "ward.D", "complete", "average", "single", "mcquitty", "median", "centroid"), or "mmm" for Mixed Markov Model clustering. Default: "pam".
dissimilarity	Character. Distance metric for sequence clustering. Only valid when cluster_by != "mmm". Default: "hamming".
...	Routed to two stages. For distance clustering (cluster_by != "mmm"), build_clusters arguments na_syms, weighted, lambda, seed, q, p, and covariates are intercepted and forwarded to the clusterer; recognised network-estimation arguments flow to build_network . Unknown arguments error before either stage is run. When cluster_by = "mmm", the recognised build_mmm arguments (n_starts, max_iter, tol, smooth, seed, covariates) are intercepted instead, and the rest flows to build_network. In both modes, when data is a netobject, its build_args are merged into the build_network side (caller's explicit values take precedence).

Details

If data is a netobject and method is not provided in ..., the original network method is inherited automatically so the per-cluster networks match the type of the input network.

Value

A netobject_group.

See Also

[build_clusters](#), [cluster_mmm](#), [build_network](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A", "B"), V2 = c("B", "C", "A", "B", "A"),
                  V3 = c("C", "A", "B", "C", "B"))
grp <- cluster_network(seqs, k = 2)
grp

set.seed(1)
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 50, TRUE), V2 = sample(LETTERS[1:4], 50, TRUE),
  V3 = sample(LETTERS[1:4], 50, TRUE), V4 = sample(LETTERS[1:4], 50, TRUE)
)
# Default: PAM clustering, relative (transition) networks
grp <- cluster_network(seqs, k = 3)

# Specify network method (cor requires numeric panel data)
## Not run:
panel <- as.data.frame(matrix(rnorm(1500), nrow = 300, ncol = 5))
grp <- cluster_network(panel, k = 2, method = "cor")

## End(Not run)

# MMM-based clustering
grp <- cluster_network(seqs, k = 2, cluster_by = "mmm")
```

cluster_summary

Cluster Summary Statistics

Description

Aggregates node-level network weights to cluster-level summaries. Computes both between-cluster transitions (how clusters connect to each other) and within-cluster transitions (how nodes connect within each cluster).

Usage

```
cluster_summary(
  x,
  clusters = NULL,
  method = c("sum", "mean", "median", "max", "min", "density", "geomean"),
  directed = TRUE,
```

```

    compute_within = TRUE
  )

```

Arguments

- | | |
|----------|---|
| x | <p>Network input. Accepts multiple formats:</p> <p>matrix Numeric adjacency/weight matrix. Row and column names are used as node labels. Values represent edge weights (e.g., transition counts, co-occurrence frequencies, or probabilities).</p> <p>netobject A network object built by build_network. Its weight matrix <code>x\$weights</code> is aggregated; a bare <code>cograph_network</code> is coerced to a <code>netobject</code> first.</p> <p>tna A <code>tna</code> object from the <code>tna</code> package. Extracts <code>x\$weights</code>.</p> <p>mcml An <code>mcml</code> object is returned unchanged.</p> |
| clusters | <p>Cluster/group assignments for nodes. Accepts multiple formats:</p> <p>NULL (default) Not usable here: <code>clusters</code> is required and <code>NULL</code> raises an error. Auto-detection of a cluster column in a <code>netobject</code>'s node table happens in build_mcml, which then calls this function with the detected assignment.</p> <p>vector Cluster membership for each node, in the same order as the matrix rows/columns. Can be numeric (1, 2, 3) or character ("A", "B"). Cluster names will be derived from unique values. Example: <code>c(1, 1, 2, 2, 3, 3)</code> assigns first two nodes to cluster 1.</p> <p>data.frame A data frame where the first column contains node names and the second column contains group/cluster names. Example: <code>data.frame(node = c("A", "B", "C"), group = c("G1", "G1", "G2"))</code></p> <p>named list Explicit mapping of cluster names to node labels. List names become cluster names, values are character vectors of node labels that must match matrix row/column names. Example: <code>list(Alpha = c("A", "B"), Beta = c("C", "D"))</code></p> |
| method | <p>Aggregation method for combining edge weights within/between clusters. Controls how multiple node-to-node edges are summarized:</p> <p>"sum" (default) Sum of all edge weights. Best for count data (e.g., transition frequencies). Preserves total flow.</p> <p>"mean" Average edge weight. Best when cluster sizes differ and you want to control for size. Note: when input is already a transition matrix (rows sum to 1), "mean" avoids size bias. Example: cluster with 5 nodes won't have 5x the weight of cluster with 1 node.</p> <p>"median" Median edge weight. Robust to outliers.</p> <p>"max" Maximum edge weight. Captures strongest connection.</p> <p>"min" Minimum edge weight. Captures weakest connection.</p> <p>"density" Sum of (non-zero) edge weights divided by the number of possible edges between the two clusters ($n_i * n_j$). Normalizes by cluster size combinations. Because zero/NA edges are stripped before aggregation, this equals "mean" exactly when the cluster-pair block is fully dense (no zero edges), and is strictly smaller than "mean" when zero edges are present (it divides by the larger possible-edge count).</p> |

	"geomean" Geometric mean of positive weights. Useful for multiplicative processes.
directed	Logical. If TRUE (default), treat network as directed. A->B and B->A are separate edges. If FALSE, edges are undirected and the matrix is symmetrized before processing.
compute_within	Logical. If TRUE (default), compute within-cluster transition matrices for each cluster. Each cluster gets its own $n_i \times n_i$ matrix showing internal node-to-node transitions. Set to FALSE to skip this computation for better performance when only between-cluster summary is needed.

Details

This is the core function for Multi-Cluster Multi-Level (MCML) analysis. Use `as_tna()` to convert results to tna objects for further analysis with the tna package.

Workflow:

Typical MCML analysis workflow:

```
# 1. Create the node-level network
net <- build_network(data, method = "relative")

# 2. Aggregate its edges to cluster level (arithmetic aggregation)
cs <- cluster_summary(net, clusters = group_assignments, method = "sum")

# 3. Read the result
print(cs)      # macro weights
summary(cs)    # one row per cluster

# 4. Promote the layers to netobjects for downstream verbs
nets <- as_tna(cs)
```

Between-Cluster Matrix Structure:

The `macro$weights` matrix has clusters as both rows and columns:

- Off-diagonal (row i , col j): Aggregated weight from cluster i to cluster j
- Diagonal (row i , col i): Within-cluster total (aggregation of internal edges)

Rows are NOT normalized. Entries are elementwise aggregates produced by method. If the caller wants probabilities, they should normalize downstream (e.g. via `as_tna()`). Mixing an arithmetic aggregation with row-normalization here (the old type = "tna" combined with `method = "min"` / `"mean"` etc.) produces numbers that sum to 1 per row but are not a probability distribution over any process; that silently-wrong combination is why type was removed from the matrix path. The sequence and edgelist paths of `build_mcml()` keep type, where the aggregation is always counts and the post-processing chooses between well-defined network constructions.

Choosing method:

Input data	Recommended method	Reason
Edge counts	"sum"	Preserves total flow between clusters
Transition matrix	"mean"	Avoids cluster size bias

Correlation matrix	"mean"	Average correlations
Dense weighted	"max" / "median"	Robust summary

Value

An `mcml` object (S3 class): a list with

macro An `mcml_layer` holding the cluster-level network: `$weights`, the $k \times k$ matrix whose entry (i, j) is the aggregation (per method) of all edges from nodes in cluster i to nodes in cluster j , with the diagonal holding the within-cluster edges – pure arithmetic, no row normalization; `$inits`, the length- k column sums of that matrix normalized to sum to 1; `$labels`, the cluster names; and `$data`, NULL on this path.

clusters Named list with one `mcml_layer` per cluster. Its `$weights` is the $n_i \times n_i$ submatrix of the nodes in that cluster and its `$inits` the normalized column sums of that submatrix. NULL when `compute_within = FALSE`.

cluster_members Named list mapping cluster names to their member node labels, e.g. `list(A = c("n1", "n2"), B = c("n3", "n4", "n5"))`.

edges NULL on this path – a matrix carries no node-level transitions. The sequence and edge-list paths of `build_mcml` fill in a tidy edge table here.

meta List with type (always "aggregate" here), method, directed, `n_nodes`, `n_clusters`, `cluster_sizes` (named integer vector) and source ("matrix").

See Also

`build_mcml` to build an `mcml` from raw transitions instead of a weight matrix, `as_tna` to promote the layers to `netobjects`, `macro_network` for the cluster-level network with one cluster expanded back into its member states

Examples

```
# -----
# Basic usage with matrix and cluster vector
# -----
set.seed(1)
mat <- matrix(runif(100), 10, 10)
rownames(mat) <- colnames(mat) <- LETTERS[1:10]

cs <- cluster_summary(mat, c(1, 1, 1, 2, 2, 2, 3, 3, 3, 3))
cs          # cluster-level (macro) weights
summary(cs) # one row per cluster

# -----
# Named list clusters (more readable)
# -----
clusters <- list(
  Alpha = c("A", "B", "C"),
  Beta  = c("D", "E", "F"),
  Gamma = c("G", "H", "I", "J")
)
```

```

cluster_summary(mat, clusters)

# -----
# A netobject as input: its weight matrix is aggregated
# -----
seqs <- data.frame(
  T1 = c("A", "C", "B", "D"), T2 = c("B", "D", "A", "C"),
  T3 = c("C", "A", "D", "B")
)
net <- build_network(seqs, method = "relative")
cluster_summary(net, list(G1 = c("A", "B"), G2 = c("C", "D")))

# -----
# Different aggregation methods
# -----
summary(cluster_summary(mat, clusters, method = "sum")) # total flow
summary(cluster_summary(mat, clusters, method = "mean")) # average
summary(cluster_summary(mat, clusters, method = "max")) # strongest

# -----
# Skip within-cluster computation for speed
# -----
cluster_summary(mat, clusters, compute_within = FALSE)

# -----
# Promote the layers to netobjects
# (as_tna() stores the aggregated weights as they are)
# -----
as_tna(cluster_summary(mat, clusters, method = "sum"))

```

coefs

Tidy coefficients from a fitted mlvar model

Description

Generic accessor for the tidy coefficient table stored on a `build_mlvar()` result. Returns a `data.frame` with one row per (outcome, predictor) pair and columns outcome, predictor, beta, se, t, p, ci_lower, ci_upper, significant.

Usage

```

coefs(x, ...)

## S3 method for class 'net_mlvar'
coefs(x, ...)

## Default S3 method:
coefs(x, ...)

```

Arguments

`x` A fitted model object - currently only `net_mvlar` is supported.

`...` Unused.

Details

Only the within-person (temporal) coefficients are tabulated - these are the lagged fixed effects that populate `fit$temporal`. The between-subjects effects that go into `fit$between` are handled via the D (I - Gamma) transformation and are not exposed as a separate tidy table.

Value

A tidy data.frame of coefficient estimates.

Examples

```
# A three-variable ESM panel: 20 people x 20 beeps. `tired` is driven by
# `happy` one beep earlier, so the temporal network should recover it.
if (requireNamespace("lme4", quietly = TRUE)) {
  set.seed(1)
  n_beep <- 20
  ar1 <- function(n, phi) as.numeric(stats::filter(stats::rnorm(n), phi,
                                                    method = "recursive"))
  panel <- do.call(rbind, lapply(seq_len(20), function(i) {
    happy <- ar1(n_beep, 0.4)
    data.frame(
      id    = i,
      beep  = seq_len(n_beep),
      happy = happy + stats::rnorm(1),
      calm  = ar1(n_beep, 0.3) + stats::rnorm(1),
      tired = 0.5 * c(0, happy[-n_beep]) + stats::rnorm(n_beep) +
        stats::rnorm(1)
    )
  }))
  fit <- build_mvlar(panel, vars = c("happy", "calm", "tired"),
                    id = "id", beep = "beep")

  fit
  coefs(fit)
  summary(fit)
}
```

compare_mmm

Compare MMM fits across different k

Description

Compare MMM fits across different k

Usage

```
compare_mmm(data, k = 2:5, return_fits = FALSE, ...)
```

```
## S3 method for class 'mmm_compare'
print(x, ...)
```

```
## S3 method for class 'mmm_compare'
summary(object, ...)
```

```
## S3 method for class 'mmm_compare'
plot(x, ...)
```

Arguments

<code>data</code>	Data frame, netobject, or tna model.
<code>k</code>	Integer vector of component counts. Values must be whole finite numbers ≥ 2 . Default: 2:5.
<code>return_fits</code>	Logical. When TRUE the fitted models are retained on the result via <code>attr(result, "fits")</code> (a list of <code>net_mmm</code> objects, named by <code>k</code>), so the user can pick the chosen model without re-running the EM. Default FALSE keeps the historical lightweight return shape – only the comparison table is allocated.
<code>...</code>	Arguments passed to <code>build_mmm</code> . In <code>plot.mmm_compare()</code> , <code>print.mmm_compare()</code> and <code>summary.mmm_compare()</code> : Unsupported. Supplying unused arguments raises an error.
<code>x</code>	For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>mmm_compare</code> .
<code>object</code>	For the <code>summary()</code> method: an object of class <code>mmm_compare</code> .

Value

A `mmm_compare` data frame, one row per requested `k`, with columns `k`, `log_likelihood`, `AIC`, `BIC`, `ICL`, `AvePP`, `Entropy` and `converged`. When `return_fits = TRUE`, the fitted `net_mmm` models are attached as `attr(result, "fits")`.

In `print.mmm_compare()`: The comparison table, invisibly, with the printed best marker column ("`<-- BIC`" / "`<-- ICL`") added.

In `summary.mmm_compare()`: A tidy data frame with one row per `k`, plus a best character column flagging the minimum-BIC and minimum-ICL solutions.

In `plot.mmm_compare()`: A `ggplot` object, invisibly.

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                   V2 = sample(c("A", "B", "C"), 30, TRUE))
comp <- compare_mmm(seqs, k = 2:3, n_starts = 1, max_iter = 10, seed = 1)
comp

seqs <- data.frame(
  V1 = sample(LETTERS[1:3], 30, TRUE), V2 = sample(LETTERS[1:3], 30, TRUE),
```

```

V3 = sample(LETTERS[1:3], 30, TRUE), V4 = sample(LETTERS[1:3], 30, TRUE)
)
comp <- compare_mmm(seqs, k = 2:3, seed = 42)
print(comp)

# Retain the fits so the chosen model needs no re-run; summary() marks
# the minimum-BIC and minimum-ICL rows in its `best` column.
comp_with_fits <- compare_mmm(seqs, k = 2:3, seed = 42, return_fits = TRUE)
summary(comp_with_fits)

```

compare_model	<i>Compare two networks descriptively</i>
---------------	---

Description

Computes a battery of descriptive comparison metrics between two networks or two weight matrices: weight deviations (mean / median / RMS / max absolute difference, relative mean absolute difference, coefficient-of- variation ratio), four correlation measures (Pearson, Spearman, Kendall, distance correlation), five dissimilarity measures (Euclidean, Manhattan, Canberra, Bray-Curtis, Frobenius), five similarity measures (Cosine, Jaccard, Dice, Overlap, RV), pattern agreements, and side-by-side network metrics. Optionally adds centrality differences and centrality correlations.

Usage

```

compare_model(x, ...)

## S3 method for class 'netobject'
compare_model(
  x,
  y,
  scaling = "none",
  measures = character(0),
  network = TRUE,
  ...
)

## S3 method for class 'cograph_network'
compare_model(
  x,
  y,
  scaling = "none",
  measures = character(0),
  network = TRUE,
  ...
)

```

```

## S3 method for class 'matrix'
compare_model(
  x,
  y,
  scaling = "none",
  measures = character(0),
  network = TRUE,
  ...
)

## S3 method for class 'netobject_group'
compare_model(
  x,
  i = 1L,
  j = 2L,
  scaling = "none",
  measures = character(0),
  network = TRUE,
  ...
)

## S3 method for class 'net_comparison'
print(x, ...)

## S3 method for class 'net_comparison'
plot(
  x,
  type = c("scatter", "heatmap", "diff_hist", "weight_dist", "all"),
  combined = TRUE,
  ...
)

```

Arguments

<code>x</code>	A netobject, cograph_network, or numeric square matrix, or a netobject_group whose members <code>i</code> and <code>j</code> are compared. For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_comparison</code> .
<code>...</code>	Ignored. In <code>plot.net_comparison()</code> and <code>print.net_comparison()</code> : Ignored.
<code>y</code>	A netobject, cograph_network, or numeric square matrix.
<code>scaling</code>	Scaling applied to both weight matrices before comparison. One of: "none" Identity (default). "minmax" $(w - \min)/(\max - \min)$; maps to $[0, 1]$. "max" $w / \max(w)$; preserves sign. "rank" Min-max of average ranks; ordinal scaling. "zscore" $(w - \bar{w})/s_w$; standard score. "robust" $(w - \text{med}(w))/\text{mad}(w)$; Huber-style robust z-score, resists outliers. "log" $\log(w)$; requires $w > 0$.

	"log1p" $\log(1 + w)$; admits $w \geq 0$. "softmax" Numerically stable softmax over the flattened vector. "quantile" Empirical CDF of the flattened vector. "frobenius" Divide the matrix by its Frobenius norm $\ W\ _F = \sqrt{\sum w_{ij}^2}$; matrix-level normalisation. "row" Row-stochastic normalisation (each row's absolute values sum to 1). Only meaningful for non-negative matrices; rows summing to zero are left unchanged. Scalings that produce negative weights (zscore, robust) are compatible with <code>network = TRUE</code> because the side-by-side metrics use Nestimate's base-R Floyd-Warshall, which handles negative weights.
measures	Character vector of centrality measures to compare. Empty by default (no centrality block). Any built-in measure is valid: "OutStrength", "InStrength", "ClosenessIn", "ClosenessOut", "Closeness", "Betweenness", "BetweennessRSP", "Diffusion", "Clustering", "InCloseness", "OutCloseness". Unknown names are ignored with a warning.
network	Logical. Include side-by-side network metrics from <code>summary()</code> ? Default TRUE.
i, j	For a <code>netobject_group</code> : index or name of the two member networks to compare. Defaults 1L and 2L.
type	Character. One of "scatter" (default - edge-weight scatter with OLS fit and correlation overlay), "heatmap" (n by n grid of x - y differences using the diverging palette), "diff_hist" (histogram of x - y absolute differences with rug + density), "weight_dist" (overlaid distributions of x and y edge weights), or "all" (2 by 2 grid of all four panels; requires the <code>gridExtra</code> package).
combined	When <code>type = "all"</code> and <code>combined = TRUE</code> (default), the four panels are stitched into a 2x2 gtable. When FALSE, returns a named list of the four ggplots so each can be printed, saved, or re-laid-out independently. Ignored for other type values.

Details

Mirrors `tna::compare()` numerically. Inputs are converted to weight matrices and scaled before comparison; the choice of scaling determines how weights from different estimators are placed on a common footing.

Value

A `net_comparison` object: a named list with `matrices`, `difference_matrix`, `edge_metrics`, `summary_metrics`, optionally `network_metrics`, `centrality_differences`, `centrality_correlations`.

In `print.net_comparison()`: x, invisibly.

In `plot.net_comparison()`: A ggplot object; for `type = "all"` with `combined = TRUE` a gtable arranged 2 by 2; for `type = "all"` with `combined = FALSE` a named list of four ggplots.

Methods

- `compare_model.netobject_group()`: Selects two members of a `netobject_group` (by index or name) and dispatches to `compare_model.netobject()`. See also [compare_networks](#), the N-way successor with tidy tables and a `plot()` that draws one view per call.
- `plot.net_comparison()`: Visualises a `net_comparison` object. Currently supports the edge-weight scatterplot (default), with the diagonal reference (perfect agreement) and the OLS regression line annotated by Pearson, Spearman, and Kendall correlations.

Examples

```
nets <- build_network(group_regulation_long, method = "relative",
                     actor = "Actor", action = "Action", time = "Time",
                     group = "Achiever")
compare_model(nets)
```

compare_networks	<i>Compare two or more networks</i>
------------------	-------------------------------------

Description

Compares any number of networks pairwise (all pairs, or every network against one reference) and returns one tidy object: an edge table, a node (centrality) table, a global-metric table and per-network structural metrics, each returned by a named verb and carrying a `pair` column so nothing ever needs list indexing. Descriptive by default; `test =` adds permutation, Bayesian and bootstrap evidence to the same tables. `plot()` draws one view per call.

Usage

```
compare_networks(
  ...,
  reference = NULL,
  scaling = c("none", "minmax", "max", "rank", "zscore", "robust", "log", "log1p",
             "softmax", "quantile", "frobenius", "row"),
  measures = c("InStrength", "OutStrength", "Betweenness"),
  labels = NULL,
  test = "none",
  iter = 1000L,
  alpha = 0.05,
  adjust = "none",
  paired = FALSE,
  rope = NULL,
  seed = NULL,
  actor = NULL
)

## S3 method for class 'net_network_comparison'
print(x, digits = 2L, ...)
```

```
## S3 method for class 'net_network_comparison'
plot(
  x,
  type = c("networks", "difference", "edges", "nodes", "global", "heatmap", "scatter",
    "inference"),
  pair = NULL,
  combined = TRUE,
  top_n = 20L,
  measure = NULL,
  labels = TRUE,
  digits = 2L,
  what = NULL,
  ...
)
```

Arguments

...	For <code>compare_networks()</code> : two or more networks, in any mix of: <code>netobject</code> , <code>netobject_group</code> (members are flattened and keep their names), <code>cograph_network</code> / <code>psychnet</code> , <code>mcml</code> , <code>tna</code> , <code>group_tna</code> , square numeric matrices, or one unnamed list of these. Name the arguments to name the networks (<code>compare_networks(early = a, late = b)</code>). For <code>plot()</code> : passed to <code>cograph::splot()</code> for the network views (e.g. <code>layout</code> , <code>node_size</code> , <code>minimum</code>); ignored by the other views and by <code>print()</code> .
reference	NULL (default) compares all pairs. A single network name or index compares every other network against that one; the reference is always <code>network_a</code> , so <code>diff = reference - other</code> .
scaling	Scaling applied to every network before comparison; one of "none" (default), "minmax", "max", "rank", "zscore", "robust", "log", "log1p", "softmax", "quantile", "frobenius", "row". Inference (<code>test != "none"</code>) requires "none": the tests are defined on the networks as estimated.
measures	Centrality measures for the node table. Any of <code>OutStrength</code> , <code>InStrength</code> , <code>ClosenessIn</code> , <code>ClosenessOut</code> , <code>Closeness</code> , <code>Betweenness</code> , <code>BetweennessRSP</code> , <code>Diffusion</code> , <code>Clustering</code> ; "all" selects those nine. NULL or <code>character(0)</code> skips the node table. Unknown names are dropped with a warning.
labels	For <code>compare_networks()</code> : optional character vector naming the networks (one per network after flattening groups); overrides argument names. For <code>plot()</code> : logical; print the signed difference on the edge and node views, default TRUE (the heatmap always shows its values).
test	Character vector of inference backends, any of "none" (default), "permutation", "bayes", "bootstrap". Several may be combined; each fills the columns it supports (see Details).
iter	Number of permutations / posterior draws / bootstrap replicates per pair. Default 1000.
alpha	Significance level (permutation, bootstrap) and 1 - ci (Bayesian credible interval). Default 0.05.

adjust	Multiplicity adjustment for permutation p-values, passed to <code>stats::p.adjust()</code> within each pair and table. Default "none".
paired	Logical; paired permutation (equal observation counts).
rope	Optional half-width of a region of practical equivalence on the difference scale (Bayesian backend only). Adds <code>bayes_p_rope</code> and a <code>bayes_decision</code> of "different", "equivalent" or "undecided".
seed	Optional integer seed. Each pair uses <code>seed + pair index</code> , so results are reproducible and independent of pair order.
actor	Optional column name identifying the actor each sequence belongs to (e.g. "student_id" when sessions are nested in students, "Group" for students nested in teams). Passed to <code>permutation()</code> , which then reassigns whole actors; see its sections <i>Nested data and actor</i> and <i>ICC and design effect</i> . Requires <code>test = "permutation"</code> . The ICC and design effects are added to global under the category "Nesting". Default NULL.
x	For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_network_comparison</code> .
digits	In <code>plot.net_network_comparison()</code> : Decimals in printed values. Default 2. In <code>print.net_network_comparison()</code> : Decimals shown. Default 2.
type	For <code>plot()</code> : one view per call. "networks" (default) draws each network once with <code>cograph::splot()</code> ; "difference" draws the signed difference network of each pair; "edges" is a ranked dumbbell of the largest edge differences; "nodes" the same for centralities; "global" the 22 comparison metrics; "heatmap" the signed difference matrix; "scatter" weight against weight; "inference" is a forest of the edge differences on the difference scale, with credible intervals when the Bayesian backend ran and the p-value printed per edge. It needs <code>test != "none"</code> and raises <code>nestimate_compare_no_test</code> otherwise.
pair	Optional selection of comparisons to draw; default all. Any of: the pair name(s) as printed ("A vs B"); the two network names (<code>c("A", "B")</code> , either order); one network name ("A", every pair it takes part in); or index/indices into the pair table (<code>1, c(1, 3)</code>). For <code>type = "networks"</code> this selects the networks taking part in the chosen pairs.
combined	When TRUE (default), a multi-pair view is one figure (facets for the ggplot views, one base-graphics page for "networks" and "difference"). When FALSE, the view is split: the ggplot views return a named list of single-pair plots, one per pair, and the base-graphics views draw one panel per page.
top_n	Number of edges shown in the edge view (largest absolute differences first). Default 20.
measure	Optional centrality measure(s) to restrict the node view.
what	Deprecated alias for <code>type</code> , kept so existing calls keep working.

Details

Guarding. Ratios are never Inf/NaN: `ratio` is NA when `weight_b == 0`, `rel_diff` is NA when both weights are 0, and `log_ratio = log1p(a) - log1p(b)` is NA when either weight is negative. No pseudo-counts are added.

Cells. Every cell of the weight matrix is a row, including the diagonal and edges absent from one network (weight 0); when both networks are undirected only from $\leq$ to cells are kept.

Inference. "permutation" (via [permutation\(\)](#)) adds perm_effect, perm_p, perm_sig to edges and nodes, and two rows M (sum of absolute edge differences) and S (largest absolute edge difference) to global with permutation p-values; with actor, the reassignment moves whole actors and global also gains the rows ICC, Design effect (edges) and Design effect (M). "bayes" (via [bayes_compare\(\)](#)) adds bayes_diff (posterior mean difference), bayes_ci_lower, bayes_ci_upper, bayes_pd (probability of direction), bayes_p, bayes_sig to edges; with rope, bayes_p_rope (normal approximation from the posterior mean and SD) and bayes_decision. "bootstrap" (via [vertex_compare\(\)](#)) appends structural rows (density, mean weight, centralization, reciprocity) to global with boot_se, boot_ci_lower, boot_ci_upper, boot_z, boot_p, boot_sig. Unified sig and evidence columns take the permutation result when run (on both edges and nodes), else the Bayesian one (on edges only – the Bayesian backend is edge-level). Permutation and Bayesian tests need networks that carry their data ([build_network\(\)](#) output, or tna objects, which are rebuilt); plain matrices support "bootstrap" only.

Value

An object of class `net_network_comparison`: a list with

- `networks`: named list of the input networks as `netobjects` (scaled weights);
- `matrices`: named list of scaled weight matrices;
- `pairs`: `data.frame`, one row per comparison: `pair`, `network_a`, `network_b`;
- `edges`: `data.frame`, one row per pair x cell: `pair`, `network_a`, `network_b`, `from`, `to`, `weight_a`, `weight_b`, `diff`, `abs_diff`, `rel_diff`, `ratio`, `log_ratio`, `rank_a`, `rank_b`, `rank_diff`, `percentile_diff`, `status` (both/only_a/only_b/neither), `higher` (`network_a`, `network_b` or equal), plus inference columns;
- `nodes`: `data.frame` (or `NULL`), one row per pair x node x measure: `pair`, `network_a`, `network_b`, `node`, `measure`, `value_a`, `value_b`, `diff`, `abs_diff`, `rank_a`, `rank_b`, `higher`, plus inference columns;
- `global`: `data.frame`, one row per pair x metric (22 descriptive metrics in five categories, plus inference rows): `pair`, `network_a`, `network_b`, `category`, `metric`, `key`, `value`, plus inference columns;
- `network_metrics`: `data.frame`, one row per network x structural metric: `network`, `metric`, `value`;
- `differences`: named list of `netdifference` objects (one per pair);
- `scaling`, `reference`, `measures`, `test`, `iter`, `alpha`, `adjust`, `paired`, `rope`, `directed` (named logical), `n_networks`, `n_pairs`.

`summary()` returns the one-row-per-pair overview table; the full tables come from the named verbs [edge_differences\(\)](#), [node_differences\(\)](#), [global_differences\(\)](#) and [network_metrics\(\)](#). `plot()` draws one view per call.

In `plot.net_network_comparison()`: `plot()` returns a `ggplot` for `type = "edges"`, `"nodes"`, `"global"`, `"heatmap"`, `"scatter"` and `"inference"`, or a named list of such plots (one per pair) when `combined = FALSE`. `type = "networks"` and `type = "difference"` draw in base graphics – with `cograph::splot()` when `cograph` is installed, otherwise with a built-in circular drawer – and return `NULL` invisibly.

In `print.net_network_comparison()`: `print()` returns `x` invisibly.

Errors

Classed conditions (nestimate_compare_*): too_few, bad_input, dim_mismatch, node_mismatch, na_weights, reference_unknown, labels_length, scaling_domain, scaling_inference, test_unsupported, unknown_pair, no_nodes, no_test (plot(type = "inference") under test = "none"), unknown_measure (selecting a measure the object does not carry); warning unknown_measure (an unknown name in measures).

Reading the figures

One colour contract in every view and every backend: "#4A6FE3" marks network_a (the reference, when one is set) as the higher of the two, "#D33F6A" marks network_b, and grey marks no difference; the plotted quantity is always $\text{diff} = a - b$. Colour never carries the sign alone – a solid line and a circular marker repeat "a higher", a dashed line and a square marker repeat "b higher", and the printed value carries its sign. When test was run, evidence is shown by opacity and a starred (edge and node views) or annotated (inference view) label: a non-significant difference is faded, never deleted.

See Also

[compare_model\(\)](#) (two-network predecessor), [permutation\(\)](#), [bayes_compare\(\)](#), [vertex_compare\(\)](#), [subtract_networks\(\)](#).

Examples

```
# Regulation networks for the three courses, compared pairwise.
courses <- build_network(group_regulation_long, method = "relative",
                        actor = "Actor", action = "Action", time = "Time",
                        group = "Course")

cmp <- compare_networks(courses)
cmp
summary(cmp)
# `pair` takes the two network names, in either order.
edge_differences(cmp, pair = c("A", "B"))
global_differences(cmp, pair = c("A", "B"))
plot(cmp) # each network once
plot(cmp, type = "difference") # signed difference per pair
plot(cmp, type = "edges", pair = c("A", "B"))

# High against low achievers, with a permutation test on every edge.
achievers <- build_network(group_regulation_long, method = "relative",
                        actor = "Actor", action = "Action",
                        time = "Time", group = "Achiever")
cmp_perm <- compare_networks(achievers, test = "permutation",
                        iter = 100, seed = 1)

summary(cmp_perm)
plot(cmp_perm, type = "inference", top_n = 10)
```

comparison_tables	<i>Tables of a network comparison</i>
-------------------	---------------------------------------

Description

Named accessors for the tables inside a `net_network_comparison` object (from `compare_networks()`). Each returns a plain data frame with one row per unit and no row names. Numeric columns are rounded to digits decimals; p-value columns are never rounded.

Usage

```
## S3 method for class 'net_network_comparison'
summary(object, pair = NULL, digits = 2L, ...)

edge_differences(x, pair = NULL, digits = 2L)

node_differences(x, pair = NULL, measure = NULL, digits = 2L)

global_differences(x, pair = NULL, digits = 2L)

network_metrics(x, digits = 2L)

## S3 method for class 'net_table'
print(x, digits = attr(x, "digits") %||% 2L, ...)
```

Arguments

<code>pair</code>	Optional selection of comparisons. Any of: the pair name(s) as printed ("A vs B"); the two network names (<code>c("A", "B")</code> , either order); one network name ("A", every pair it takes part in); or index/indices into the pair table (1, <code>c(1, 3)</code>). NULL keeps all.
<code>digits</code>	Decimals kept in numeric columns. Default 2.
<code>...</code>	Ignored.
<code>x, object</code>	A <code>net_network_comparison</code> object (for <code>print.net_table()</code> , a table returned by one of these verbs).
<code>measure</code>	Optional centrality measure name(s) to keep.

Value

- `summary()`: one row per pair – pair, network_a, network_b, n_cells, n_differing, share_higher_a, share_higher_b, mean_abs_diff, max_abs_diff, pearson, spearman, cosine, jaccard, top_edge, top_edge_higher, and with inference n_sig_edges, n_sig_nodes, m_stat, m_p, s_stat, s_p. The per-edge, per-node, per-metric and per-network tables are the four verbs below.

- `edge_differences()`: one row per pair x transition: pair, network_a, network_b, from, to, weight_a, weight_b, diff, abs_diff, rel_diff, ratio, log_ratio, rank_a, rank_b, rank_diff, percentile_diff, status, higher, and the inference columns when test was used.
- `node_differences()`: one row per pair x state x measure: pair, network_a, network_b, node, measure, value_a, value_b, diff, abs_diff, rank_a, rank_b, higher, plus inference columns. Errors (class `nestimate_compare_no_nodes`) when `compare_networks()` was called with no measures.
- `global_differences()`: one row per pair x metric: pair, network_a, network_b, category, metric, key, value, plus inference rows and columns.
- `network_metrics()`: one row per network x structural metric: network, metric, value.

Every table is a data frame of class `net_table` whose `print()` shows whole numbers without decimals, exact zeros as 0, other values with digits decimals, and p-values with three decimals; `print()` itself returns the table invisibly.

Examples

```

achievers <- build_network(group_regulation_long, method = "relative",
                           actor = "Actor", action = "Action",
                           time = "Time", group = "Achiever")
cmp <- compare_networks(achievers)
summary(cmp)
edge_differences(cmp)
edge_differences(cmp, digits = 4)
node_differences(cmp, measure = "InStrength")
global_differences(cmp)
network_metrics(cmp)

```

composites

Cluster Scores From a Psychometric MCML Fit

Description

The per-observation cluster scores that a re-estimated `build_mcml_pc` macro network was fitted on: each respondent's weighted, sign-corrected score on every cluster. These are the scores to carry into a profile analysis, a regression, or any downstream model that needs one number per cluster per respondent.

Usage

```

composites(x, ...)

## S3 method for class 'mcml_pc'
composites(x, ...)

```

Arguments

`x` An object carrying cluster scores.

`...` Ignored.

Value

A data frame with one row per row of the input data, in input order and with the input's row names, and one numeric column per cluster, named by the cluster. A row whose members of a cluster are all missing is NA in that column (items missing only in part are averaged over the observed ones). The macro network is estimated on the complete rows, so `build_network(composites(fit), method = ...)` reproduces it.

The `mcml_pc` method errors with class `"nestimate_no_composites"` for the descriptive aggregations (`"average"`, `"escoufier"`, `"cancor"`), which relate clusters without ever forming a score.

See Also

[build_mcml_pc](#) to create the fit, [item_loadings](#) for the item weights behind these scores.

Examples

```
set.seed(1)
f <- stats::rnorm(200)
g <- stats::rnorm(200)
df <- data.frame(a1 = f + stats::rnorm(200), a2 = f + stats::rnorm(200),
                 a3 = f + stats::rnorm(200), b1 = g + stats::rnorm(200),
                 b2 = g + stats::rnorm(200), b3 = g + stats::rnorm(200))
clusters <- list(A = c("a1", "a2", "a3"), B = c("b1", "b2", "b3"))
fit <- build_mcml_pc(df, clusters, aggregation = "loadings",
                    method = "cor")
head(composites(fit))
```

convert_sequence_format

Convert Sequence Data to Different Formats

Description

Convert wide or long sequence data into frequency counts, one-hot encoding, edge lists, or follows format.

Usage

```
convert_sequence_format(
  data,
  seq_cols = NULL,
  id_col = NULL,
```

```

    action = NULL,
    time = NULL,
    format = c("frequency", "onehot", "edgelist", "follows")
  )

```

Arguments

<code>data</code>	Data frame containing sequence data.
<code>seq_cols</code>	Character vector. Names of columns containing sequential states (for wide format input). If NULL, all columns except <code>id_col</code> are used. Default: NULL.
<code>id_col</code>	Character vector. Name(s) of the ID column(s). For long format, required. For wide format, optional: if NULL, the first column is used as the id only when it is a genuine identifier (its values are disjoint from the states in the remaining columns); for canonical wide sequence data with no id column (e.g. $V_1 \dots V_n / T_1 \dots T_n$), a row-index id is synthesized and every column is treated as a sequence column. Default: NULL.
<code>action</code>	Character or NULL. Name of the column containing actions/states (for long format input). If provided, data is treated as long format. Default: NULL.
<code>time</code>	Character or NULL. Name of the time column for ordering actions within sequences (for long format). Default: NULL.
<code>format</code>	Character. Output format: "frequency" Count of each action per sequence (wide, one column per state). "onehot" Binary presence/absence of each action per sequence. "edgelist" Consecutive transition pairs (from, to) per sequence. "follows" Each action paired with the action that preceded it.

Value

A data frame in the requested format:

frequency ID columns + one integer column per state with counts.

onehot ID columns + one binary column per state (0/1).

edgelist ID columns + from and to columns.

follows ID columns + act and follows columns.

See Also

[frequencies](#) for building transition frequency matrices.

Examples

```

# Wide format input
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
convert_sequence_format(seqs, format = "frequency")
convert_sequence_format(seqs, format = "edgelist")

```

cooccurrence

*Build a Co-occurrence Network***Description**

Constructs an undirected co-occurrence network from various input formats. Entities that appear together in the same transaction, document, or record are connected, with edge weights reflecting raw counts or a similarity measure. Argument names follow the citenets convention.

Usage

```
cooccurrence(
  data,
  field = NULL,
  by = NULL,
  sep = NULL,
  similarity = c("none", "jaccard", "cosine", "inclusion", "association", "dice",
    "equivalence", "relative"),
  threshold = 0,
  min_occur = 1L,
  diagonal = TRUE,
  top_n = NULL,
  ...
)
```

Arguments

data	Input data. Accepts: • A data.frame with a delimited column (field + sep). • A data.frame in long/bipartite format (field + by). • A binary (0/1) data.frame or matrix (auto-detected). • A wide sequence data.frame or matrix (non-binary). • A list of character vectors (each element is a transaction).
field	Character. The entity column - determines what the nodes are. For delimited format, a single column whose values are split by sep. For long/bipartite format, the item column. For multi-column delimited, a vector of column names whose split values are pooled per row.
by	Character or NULL. What links the nodes. For long/bipartite format, the grouping column (e.g., "paper_id", "session_id"). Each unique value of by defines one transaction. If NULL (default), entities co-occur within the same row/document.
sep	Character or NULL. Separator for splitting delimited fields (e.g., ";", " , "). Default NULL.
similarity	Character. Similarity measure applied to the raw co-occurrence counts. One of: "none" Raw co-occurrence counts.

	"jaccard" $C_{ij}/(f_i + f_j - C_{ij})$.
	"cosine" $C_{ij}/\sqrt{f_i \cdot f_j}$ (Salton's cosine).
	"inclusion" $C_{ij}/\min(f_i, f_j)$ (Simpson coefficient).
	"association" $C_{ij}/(f_i \cdot f_j)$ (association strength / probabilistic affinity index; van Eck & Waltman, 2009).
	"dice" $2C_{ij}/(f_i + f_j)$.
	"equivalence" $C_{ij}^2/(f_i \cdot f_j)$ (Salton's cosine squared).
	"relative" Row-normalized: each row sums to 1.
threshold	Numeric. Minimum edge weight to retain. Edges below this value are set to zero. Applied <i>after</i> similarity normalization. Default 0.
min_occur	Integer. Minimum entity frequency (number of transactions an entity must appear in). Entities below this threshold are dropped before computing co-occurrence. Default 1 (keep all).
diagonal	Logical. If TRUE (default), the diagonal of the co-occurrence matrix is kept (item self-co-occurrence = item frequency). If FALSE, the diagonal is zeroed.
top_n	Integer or NULL. If specified, return only the top top_n edges by weight. Default NULL (all edges).
...	Currently unused.

Details

Six input formats are supported, auto-detected from the combination of field, by, and sep:

1. **Delimited:** field + sep (single column). Each cell is split by sep, trimmed, and de-duplicated per row.
2. **Multi-column delimited:** field (vector) + sep. Values from multiple columns are split, pooled, and de-duplicated per row.
3. **Long bipartite:** field + by. Groups by by; unique values of field within each group form a transaction.
4. **Binary matrix:** No field/by/sep, all values 0/1. Columns are items, rows are transactions.
5. **Wide sequence:** No field/by/sep, non-binary. Unique values across each row form a transaction.
6. **List:** A plain list of character vectors.

The pipeline converts all formats into a list of character vectors (transactions), optionally filters by min_occur, builds a binary transaction matrix, computes `crossprod(B)` for the raw co-occurrence counts, normalizes via the chosen similarity, then applies threshold and top_n filtering.

Value

A `netobject` (undirected, class `c("netobject", "cograph_network")`) with `method = "co_occurrence_fn"` and `$data = NULL` - this function does *not* go through `build_network`, so the result carries no source data and the data-resampling verbs (`bootstrap_network`, `permutation`) cannot be run on it. The `$weights` matrix holds the similarity (or raw) co-occurrence values, one row/column per retained item. The `$params` list records similarity, threshold, min_occur, diagonal, top_n, n_transactions and n_items.

References

van Eck, N. J., & Waltman, L. (2009). How to normalize co-occurrence data? An analysis of some well-known similarity measures. *Journal of the American Society for Information Science and Technology*, 60(8), 1635–1651.

See Also

[build_cna](#) for sequence-positional co-occurrence via `build_network()`.

Examples

```
# Delimited field (e.g., keyword co-occurrence)
df <- data.frame(
  id = 1:4,
  keywords = c("network; graph", "graph; matrix; network",
               "matrix; algebra", "network; algebra; graph")
)
net <- cooccurrence(df, field = "keywords", sep = ";")

# Long/bipartite
long_df <- data.frame(
  paper = c(1, 1, 1, 2, 2, 3, 3),
  keyword = c("network", "graph", "matrix", "graph", "algebra",
              "network", "algebra")
)
net <- cooccurrence(long_df, field = "keyword", by = "paper")

# List of transactions
transactions <- list(c("A", "B"), c("B", "C"), c("A", "B", "C"))
net <- cooccurrence(transactions, similarity = "jaccard")

# Binary matrix
bin <- matrix(c(1,0,1, 1,1,0, 0,1,1), nrow = 3, byrow = TRUE,
              dimnames = list(NULL, c("X", "Y", "Z")))
net <- cooccurrence(bin)
```

Description

Draws how state proportions (or counts) evolve across time points. For each time column, tabulates how many sequences are in each state and renders the result as a stacked area (default) or stacked bar chart. Accepts the same inputs as [sequence_plot](#).

Usage

```
distribution_plot(
  x,
  group = NULL,
  scale = c("proportion", "count"),
  geom = c("area", "bar"),
  na = TRUE,
  trim = NULL,
  trim_clusterwise = FALSE,
  state_colors = NULL,
  na_color = "grey90",
  frame = FALSE,
  width = NULL,
  height = NULL,
  main = NULL,
  show_n = TRUE,
  time_label = "Time",
  xlab = NULL,
  y_label = NULL,
  ylab = NULL,
  tick = NULL,
  ncol = NULL,
  nrow = NULL,
  combined = TRUE,
  legend = c("right", "bottom", "none"),
  legend_size = NULL,
  legend_title = NULL,
  legend_ncol = NULL,
  legend_border = NA,
  legend_bty = "n"
)
```

Arguments

<code>x</code>	Wide-format sequence data. Accepts the same inputs as sequence_plot : <code>data.frame</code> , <code>matrix</code> , <code>netobject</code> , <code>net_clustering</code> , <code>netobject_group</code> , <code>net_mmm</code> , or <code>tna</code> . When clustering info is available, one panel is drawn per cluster.
<code>group</code>	Optional grouping vector (length <code>nrow(x)</code>) producing one panel per group. <code>NULL</code> (default) falls back to the cluster assignments carried by a <code>net_clustering</code> / <code>net_mmm</code> / <code>netobject_group</code> input; supplying <code>group</code> overrides them.
<code>scale</code>	"proportion" (default) divides each column by its total so bands fill 0..1. "count" keeps raw counts.
<code>geom</code>	"area" (default) draws stacked polygons; "bar" draws stacked bars.
<code>na</code>	If <code>TRUE</code> (default), <code>NA</code> cells are shown as an extra band coloured <code>na_color</code> .
<code>trim</code>	Optional time-axis truncation, to stop a few long sequences from stretching the plot. <code>NULL</code> (default) keeps the full width. A fraction in $(0, 1)$ drops everything

	past that quantile of sequence lengths (e.g. <code>trim = 0.95</code>); a value ≥ 1 is an absolute cut (<code>trim = 50</code> keeps the first 50 time points). See sequence_plot .
<code>trim_clusterwise</code>	Grouped plots only, fractional trim only. FALSE (default) uses one pooled cutoff for every panel so the time axes stay aligned; TRUE crops each group to its own length quantile (panels can differ in width). See sequence_plot .
<code>state_colors</code>	Colours for the state fills. Either an unnamed vector, one colour per state in level order, or a named lookup (<code>c(plan = "#0072B2")</code>) where only the states you name are overridden and the rest keep the default Okabe-Ito palette. Names this plot does not draw are dropped with a message, so one palette can be reused across figures.
<code>na_color</code>	Colour for the NA band. Default "grey90".
<code>frame</code>	FALSE (default) draws no panel box; TRUE draws a box around each panel.
<code>width, height</code>	Optional device dimensions. See sequence_plot .
<code>main</code>	Plot title.
<code>show_n</code>	Append "(n = N)" (per-group when grouped) to the title.
<code>time_label</code>	X-axis label.
<code>xlab</code>	Alias for <code>time_label</code> .
<code>y_label</code>	Y-axis label. Defaults to "Proportion" or "Count" based on scale.
<code>ylab</code>	Alias for <code>y_label</code> .
<code>tick</code>	Show every Nth x-axis label. NULL = auto.
<code>ncol, nrow</code>	Facet grid dimensions. NULL = auto: <code>ncol = ceiling(sqrt(G))</code> , <code>nrow = ceiling(G / ncol)</code> . Ignored when <code>combined = FALSE</code> .
<code>combined</code>	When TRUE (default), groups are arranged on one figure via <code>graphics::layout()</code> . When FALSE, each group is drawn on its own page (one full-size figure per group, with its own legend). Useful when you want each group at full size in knitr (<code>fig.show = "asis"</code>) or to save each as a separate file. Single-group calls (<code>G == 1</code>) ignore this argument.
<code>legend</code>	Legend position: "right" (default), "bottom", or "none".
<code>legend_size</code>	Legend text size. NULL (default) auto-scales from device width (clamped to <code>[0.65, 1.2]</code>).
<code>legend_title</code>	Optional legend title.
<code>legend_ncol</code>	Number of legend columns.
<code>legend_border</code>	Swatch border colour.
<code>legend_bty</code>	"n" (borderless) or "o" (boxed).

Value

Invisibly, a list describing the drawn figure:

counts Named list, one entry per group, each a (state x time point) numeric matrix of cell counts. Rows are named by levels; columns are the retained time points.

proportions Same shape as counts, each column divided by its total.

levels Character vector of state labels in plotting order, with "NA" appended when `na = TRUE`.
palette Character vector of fill colours, parallel to `levels`.
groups Character vector of group labels ("all" when ungrouped), parallel to `counts / proportions`.

See Also

[sequence_plot](#), [build_clusters](#)

Examples

```
distribution_plot(as.data.frame(trjectories))
```

effects_table	<i>Effect Table of a Fitted Outcome Model</i>
---------------	---

Description

The tidy one-row-per-term table of estimates, confidence intervals and corrected p-values.

Usage

```
effects_table(
  x,
  intercept = FALSE,
  significant = FALSE,
  alpha = 0.05,
  digits = 3
)
```

Arguments

<code>x</code>	A <code>net_outcome_model</code> from outcome_model .
<code>intercept</code>	Keep the intercept row? Default <code>FALSE</code> .
<code>significant</code>	Keep only terms whose corrected p-value is below <code>alpha</code> ? Default <code>FALSE</code> .
<code>alpha</code>	Threshold used by <code>significant</code> . Default <code>0.05</code> .
<code>digits</code>	Rounding for the numeric columns. Default 3; <code>p_value</code> and <code>p_adj</code> are never rounded.

Value

A `data.frame` with the same columns as the model's effect table, one row per retained term. The intercept row is dropped unless `intercept = TRUE`, and every term is kept unless `significant = TRUE` restricts them to `p_adj < alpha`.

Examples

```
set.seed(1)
d <- data.frame(hint = rbinom(200, 1, 0.5))
d$success <- rbinom(200, 1, plogis(-0.3 + 0.9 * d$hint))
effects_table(outcome_model(d, outcome = "success", predictors = "hint"))
```

entropy_bayes

Bayesian Transition Entropy

Description

Bayesian estimation of the transition entropy quantities of [transition_entropy](#) and the edge-level decomposition of [entropy_network](#). Each row of the transition matrix gets an independent Dirichlet posterior (counts + prior); Monte Carlo draws propagate count uncertainty into the entropy rate, the per-state branching entropies, and every edge's entropy contribution, yielding posterior means and credible intervals.

The practical purpose is to **exclude unstable estimates**: an edge whose contribution rests on a handful of observations has a wide posterior, and is flagged non-credible unless it credibly accounts for at least `min_share` of the process entropy. `$model` is the entropy network with non-credible edges zeroed - the stable entropy skeleton.

Usage

```
entropy_bayes(
  x,
  prior = 0.5,
  draws = 4000,
  ci = 0.95,
  min_share = 0.01,
  base = 2,
  seed = NULL
)

## S3 method for class 'net_entropy_bayes'
print(x, digits = 3, ...)

## S3 method for class 'net_entropy_bayes_group'
print(x, ...)

## S3 method for class 'net_entropy_bayes'
summary(object, ...)

## S3 method for class 'net_entropy_bayes'
plot(x, top = 25, title = "Bayesian edge entropy contributions", ...)
```

Arguments

<code>x</code>	A frequency netobject (<code>build_network(method = "frequency")</code>), any netobject that carries its <code>\$data</code> (counts are rebuilt automatically), a count matrix, or a wide sequence data.frame. Group dispatch on <code>netobject_group</code> . For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_entropy_bayes</code> or <code>net_entropy_bayes_group</code> .
<code>prior</code>	Numeric. Dirichlet prior concentration added to every cell of the count matrix (default 0.5, Jeffreys).
<code>draws</code>	Integer. Number of Monte Carlo posterior draws (default 4000).
<code>ci</code>	Numeric in (0, 1). Credible interval mass (default 0.95).
<code>min_share</code>	Numeric in [0, 1). An edge is credible when the lower bound of the credible interval of its <i>share</i> of the entropy rate exceeds this value (default 0.01: the edge credibly accounts for at least 1% of process entropy).
<code>base</code>	Numeric. Logarithm base (default 2, bits).
<code>seed</code>	Integer or NULL. RNG seed for reproducibility.
<code>digits</code>	Integer. Digits to round numeric output. Default 3.
<code>...</code>	In <code>plot.net_entropy_bayes()</code> , <code>print.net_entropy_bayes()</code> , <code>print.net_entropy_bayes_group()</code> and <code>summary.net_entropy_bayes()</code> : Ignored.
<code>object</code>	For the <code>summary()</code> method: an object of class <code>net_entropy_bayes</code> .
<code>top</code>	Integer. Show at most this many edges, by posterior mean contribution (default 25).
<code>title</code>	Character. Plot title.

Details

With `prior > 0` the posterior puts mass on every transition, so contribution draws are strictly positive and a naive "CI excludes zero" rule would flag every edge as credible. The share criterion is used instead: an edge is stable when it credibly carries at least `min_share` of $h(P)$. Unobserved transitions (count 0) get only prior mass and are never credible under any sensible `min_share`.

The posterior mean entropy rate is typically slightly below the plug-in estimate on sparse data (Dirichlet smoothing pulls rows toward uniform but averages over uncertainty); the difference vanishes as counts grow.

Value

An object of class "net_entropy_bayes" with:

summary Tidy data.frame - one row per chain-level quantity (`entropy_rate`, `stationary_entropy`, `redundancy`), with posterior mean, sd, `ci_lower`, `ci_upper`.

states Tidy data.frame - one row per state: posterior mean/CI of the row entropy and of the stationary probability.

edges Tidy data.frame - one row per observed transition: posterior mean/sd/CI of the contribution (bits), posterior mean/CI of its share of the entropy rate, and the credible flag.

network netobject - posterior-mean entropy network (all edges), carrying the entropy house style.

model netobject - the pruned entropy network: posterior means where credible, 0 elsewhere.

draws_entropy_rate Numeric vector of posterior entropy-rate draws (for further analysis or plotting).

prior, draws, ci, min_share, base, states_names Call metadata.

For a netobject_group the result is a "net_entropy_bayes_group": a named list holding one such object per group.

In `print.net_entropy_bayes()` and `print.net_entropy_bayes_group()`: x invisibly.

In `summary.net_entropy_bayes()`: The tidy edge table (data.frame), one row per observed transition, sorted by posterior mean contribution, returned invisibly. The chain-level and per-edge tables are printed as a side effect.

In `plot.net_entropy_bayes()`: A ggplot object.

Methods

- `plot.net_entropy_bayes()`: Forest plot of the per-edge entropy contributions: posterior mean and credible interval, credible edges in Okabe-Ito blue, unstable (non-credible) edges in grey. The dashed line marks min_share of the posterior-mean entropy rate - the stability criterion.

References

Cover, T.M. & Thomas, J.A. (2006). *Elements of Information Theory*, 2nd ed. Wiley.

See Also

[transition_entropy](#), [entropy_network](#), [bayes_compare](#)

Examples

```
net <- build_network(group_regulation_long, method = "relative",
                    actor = "Actor", action = "Action", time = "Time")
eb <- entropy_bayes(net, draws = 1000, seed = 1)
eb
summary(eb)
plot(eb)
```

Description

Decomposes the entropy rate of a Markov transition process edge by edge and returns the decomposition as a network. The entropy rate $H = -\sum_{i,j} \pi_i P_{ij} \log P_{ij}$ ([transition_entropy](#); Krejtz et al. 2015, 2025) is an additive sum over transitions, so every edge $i \rightarrow j$ owns the exact term $\pi_i P_{ij} \log(1/P_{ij})$ of the chain-level uncertainty. **No new quantity is estimated**: the network displays the summands of the entropy-rate equation on the transition graph, locating the process's uncertainty spatially. The returned object is a regular netobject / cograph_network, so it prints, summarises, and plots (`cograph::splot()`) like any other Nestimate network.

Usage

```
entropy_network(
  x,
  base = 2,
  weight = c("contribution", "surprisal", "production"),
  scaling = c("none", "share", "chance"),
  normalize = TRUE
)
```

Arguments

x	A netobject, cograph_network, tna object, row-stochastic numeric transition matrix, or a wide sequence data.frame (rows = actors, columns = time-steps; a relative transition network is built automatically).
base	Numeric. Logarithm base. 2 (default) for bits, exp(1) for nats, 10 for hartleys.
weight	Character. Edge weight definition: "contribution" (default) $\pi_i P_{ij} \log_b(1/P_{ij})$ - the edge's additive share of the entropy rate; all edge weights sum exactly to $h(P)$. Thick edges are transitions that are both frequent <i>and</i> unpredictable. "surprisal" $\log_b(1/P_{ij})$ - the information content of observing the transition, ignoring how often it occurs. Thick edges are rare, surprising transitions. Low surprisal = expected transition; this is the edge-level predictability reading. "production" $F_{ij} \log_b(F_{ij}/F_{ji})$ with $F_{ij} = \pi_i P_{ij}$ - the edge's contribution to the entropy production rate (irreversibility). Positive on the dominant direction of a pair, negative on the reverse; the two always sum to $(F_{ij} - F_{ji}) \log_b(F_{ij}/F_{ji}) \geq 0$. Pairs with flow in only one direction have infinite production and are excluded (weight 0); their count is reported in <code>\$params\$n_oneway_pairs</code> . Total over included pairs is <code>\$params\$production_rate</code> - 0 iff the chain is reversible (detailed balance).
scaling	Character. "none" (default) keeps weights in base-units. "share" (contribution only) expresses each edge as its <i>percentage</i> of the entropy rate - weights sum to 100 and each label reads "this transition holds x% of the process's unpredictability"; comparable across networks regardless of state count or base. "chance" (surprisal only) divides by $\log_b n$ - the surprisal of a chance-level transition when all n next states are equally likely: 1 = chance level, below 1 = expected transition, above 1 = rarer than chance.

normalize Logical. If TRUE (default), rows that do not sum to 1 are normalised automatically (with a warning).

Details

Impossible transitions ($P_{ij} = 0$) get weight 0 under both definitions (the $0 \log 0 := 0$ convention), so the entropy network has the same support as the transition network. Self-loops are retained like any other edge. Deterministic transitions ($P_{ij} = 1$) also get weight 0: observing the inevitable carries no information.

Value

A netobject (also class `cograph_network`) whose `$weights` hold the per-edge entropy quantities. When `x` is a fitted network (or sequence data, from which a relative network is built), the result is that network with entropy weights swapped in - `$inits`, `$meta`, `$node_groups`, and node coordinates are inherited, so it plots with the same TNA styling and layout as its source. `$method` is "entropy". `$params` carries base, weight, scaling, entropy_rate, and the stationary distribution stationary (plus production_rate and n_oneway_pairs when weight = "production").

The object also declares the entropy house style through the `$meta$splot` producer contract (honoured by `cograph` $\geq 2.4.4$): no minimum-weight pruning (bit values are smaller than probabilities), 2-digit edge labels, vermilion edges, node rings showing the stationary distribution, and TNA rather than psychometric geometry. Because the contract states the styling outright, `cograph` needs no knowledge of the "entropy" method. `cograph::splot(ent)` therefore renders correctly with no arguments; any user argument overrides the contract.

References

- Cover, T.M. & Thomas, J.A. (2006). *Elements of Information Theory*, 2nd ed., chapter 4. Wiley.
- Krejtz, K., Duchowski, A., Szmids, T., Krejtz, I., Gonzalez Perilli, F., Pires, A., Vilaro, A., & Villalobos, N. (2015). Gaze transition entropy. *ACM Transactions on Applied Perception*, 13(1), 4:1-4:20. doi:10.1145/2834121
- Krejtz, K., Hughes, C.J., Stasiak, I., Duchowski, A., & Krejtz, I. (2025). Real-time mobile transition matrix entropy based on eye and head movements. *Proceedings of ETRA '25*. doi:10.1145/3715669.3723128

See Also

[transition_entropy](#) for the chain- and state-level summary, [entropy_bayes](#) for credible intervals on the decomposition, [build_network](#).

Examples

```
net <- build_network(group_regulation_long,
                     method = "relative",
                     actor = "Actor", action = "Action", time = "Time")
ent <- entropy_network(net)
ent

if (requireNamespace("cograph", quietly = TRUE)) {
```

```

    cograph::splot(ent)
}

```

entropy_trajectory *Sliding-Window Transition Entropy Trajectory*

Description

Tracks transition entropy over time: transitions are built within each actor's sequence, pooled in temporal order, and a window of fixed size slides across the stream, yielding one entropy estimate per window. This turns transition entropy from a snapshot into a process measure - declining entropy signals routinization, rising entropy exploration, and level shifts mark phase changes (cf. Krejtz et al., 2025, who track gaze transition entropy through task phases this way).

Usage

```

entropy_trajectory(
  data,
  action,
  actor = NULL,
  time = NULL,
  group = NULL,
  window = 500L,
  step = NULL,
  base = 2
)

## S3 method for class 'net_entropy_trajectory'
print(x, digits = 3, ...)

## S3 method for class 'net_entropy_trajectory'
summary(object, ...)

## S3 method for class 'net_entropy_trajectory'
plot(
  x,
  normalized = FALSE,
  span = 0.4,
  title = "Transition entropy over time",
  ...
)

```

Arguments

data A long-format data.frame of timestamped events.

action	Character. Name of the column holding the state/action.
actor	Character or NULL. Column identifying sequences; transitions are only formed between consecutive events of the same actor. NULL treats the data as one sequence.
time	Character or NULL. Timestamp column used to order events within actors and to place windows on a real time axis. NULL keeps the row order and uses the transition index as the axis.
group	Character or NULL. Column splitting the data into parallel trajectories (e.g. condition, achievement level).
window	Integer. Number of transitions per window (default 500).
step	Integer. Stride between window starts (default window % 5).
base	Numeric. Logarithm base (default 2, bits).
x	For the print() and plot() methods: an object of class net_entropy_trajectory.
digits	Integer. Digits to round numeric output. Default 3.
...	In plot.net_entropy_trajectory(), print.net_entropy_trajectory() and summary.net_entropy_trajectory(): Ignored.
object	For the summary() method: an object of class net_entropy_trajectory.
normalized	Logical. Plot entropy_norm instead of raw bits (default FALSE).
span	Numeric. Loess span (default 0.4).
title	Character. Plot title.

Details

Per-window entropy is the empirical conditional entropy $-\sum_{ij}(n_{ij}/N)\log_b(n_{ij}/n_{i\cdot})$ - rows weighted by observed occupancy rather than the eigenvector stationary distribution. Within a short window the chain is routinely non-ergodic (absorbing fragments, unvisited states), where the eigenvector is undefined or misleading; the empirical estimator is the standard windowed choice and converges to the stationary entropy rate for long stationary stretches.

Windows shorter than window at the tail are dropped; if the whole stream is shorter than window, one window covering everything is returned with a warning.

Value

An object of class "net_entropy_trajectory" with:

trajectory Tidy data.frame, one row per window: group, window (index), time (window midpoint; transition index when no time column), time_start, time_end, n_transitions, n_states (distinct states in the window), entropy (bits per transition) and entropy_norm (divided by $\log_b$ of the window's active-state count, floored at 2 states so the ceiling is never zero; in $[0, 1]$).

window, step, base, states Call metadata; states is the global state set.

In print.net_entropy_trajectory(): x invisibly.

In summary.net_entropy_trajectory(): Tidy per-group data.frame: windows, mean/sd/min/max entropy, entropy at the first and last window, and their difference (negative = routinization).

In plot.net_entropy_trajectory(): A ggplot object.

Methods

- `plot.net_entropy_trajectory()`: Raw per-window entropy as faint lines with a loess-smoothed trend per group, Okabe-Ito coloured. Declining trend = routinization; level shifts = phase changes.

References

- Krejtz, K., Hughes, C.J., Stasiak, I., Duchowski, A., & Krejtz, I. (2025). Real-Time Mobile Transition Matrix Entropy Based on Eye and Head Movements. *ETRA '25*. doi:10.1145/3715669.3723128
- Cover, T.M. & Thomas, J.A. (2006). *Elements of Information Theory*, 2nd ed. Wiley.

See Also

[transition_entropy](#) for the whole-process snapshot, [entropy_bayes](#) for credible intervals on it.

Examples

```
tr <- entropy_trajectory(group_regulation_long,
                        action = "Action", actor = "Actor",
                        time = "Time", group = "Achiever")

tr
summary(tr)
plot(tr)
```

estimate_network	<i>Estimate a Network (Deprecated)</i>
------------------	--

Description

This function is deprecated. Use [build_network](#) instead.

Usage

```
estimate_network(
  data,
  method = "relative",
  params = list(),
  scaling = NULL,
  threshold = 0,
  level = NULL,
  ...
)
```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance). A fitted <code>net_clustering</code> or <code>net_mmm</code> object is also accepted: the per-cluster networks are (re)built and a <code>netobject_group</code> is returned.
method	Character. Defaults to "relative" for backward compatibility.
params	Named list. Method-specific parameters passed to the estimator function (e.g. <code>list(gamma = 0.5)</code> for <code>glasso</code> , or <code>list(format = "wide")</code> for transition methods). This is the key composability feature: downstream functions like <code>bootstrap</code> or <code>grid search</code> can store and replay the full <code>params</code> list without knowing method internals. Transition estimators accept <code>tna</code> -style sequence options such as <code>weighted</code> and <code>concat</code> (and the low-level <code>begin_state</code> / <code>end_state</code> , of which <code>start</code> / <code>end</code> are the public form – see those arguments). Column-like entries in <code>params</code> (<code>action</code> , <code>id</code> , <code>id_col</code> , <code>actor</code> , <code>time</code> , <code>session</code> , <code>order</code> , <code>cols</code> , <code>codes</code> , and <code>group</code>) are resolved before format detection and must name existing columns. If the same column role is supplied both directly and through <code>params</code> , the names must agree.
scaling	Character vector or <code>NULL</code> . Post-estimation scaling to apply (in order). Options: "minmax", "max", "rank", "normalize". Can combine: <code>c("rank", "minmax")</code> . Default: <code>NULL</code> (no scaling).
threshold	Numeric. Absolute values below this are set to zero in the result matrix. Default: 0 (no thresholding).
level	Character or <code>NULL</code> . Multilevel decomposition for the undirected association methods (<code>cor</code> , <code>pcor</code> , <code>glasso</code>); a directed estimator errors. One of <code>NULL</code> , "between", "within", "both". Requires an <code>id</code> column, supplied either as <code>actor</code> or as <code>params\$id</code> / <code>params\$id_col</code> . Default: <code>NULL</code> .
...	Additional arguments passed to build_network .

Value

A `netobject` (see [build_network](#)).

See Also

[build_network](#)

Examples

```
data <- data.frame(A = c("x", "y", "z", "x"), B = c("y", "x", "z", "y"))
net <- estimate_network(data, method = "relative")
```

euler_characteristic *Euler Characteristic*

Description

Computes $\chi = \sum_{k=0}^d (-1)^k f_k$ where f_k is the number of k-simplices. By the Euler-Poincare theorem, $\chi = \sum_k (-1)^k \beta_k$.

Usage

```
euler_characteristic(sc)
```

Arguments

sc A simplicial_complex object.

Value

Integer.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
euler_characteristic(sc)
```

evaluate_links *Evaluate Link Predictions Against Known Edges*

Description

Computes AUC-ROC, precision@k, and average precision for link predictions against a set of known true edges.

Usage

```
evaluate_links(pred, true_edges, k = c(5L, 10L, 20L))
```

Arguments

pred A net_link_prediction object.

true_edges A data frame with columns from and to, or a binary matrix where 1 indicates a true edge.

k Integer vector. Values of k for precision@k. Default: c(5, 10, 20).

Value

A data frame with columns: method, auc, average_precision, and one precision_at_k column per k value.

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "C", "D", "A", "C", "E", "B"),
  V2 = c("B", "C", "D", "E", "C", "E", "A", "D"),
  V3 = c("C", "D", "E", "A", "D", "A", "B", "E")
)
net <- build_network(seqs, method = "relative")
pred <- predict_links(net, exclude_existing = FALSE)

# Evaluate against the network's own edges as the known truth
evaluate_links(pred, extract_edges(net, threshold = 0.001))
```

extract_edges	<i>Extract Edge List with Weights</i>
---------------	---------------------------------------

Description

Extract an edge list from a TNA model, representing the network as a data frame of from-to-weight tuples.

Usage

```
extract_edges(model, threshold = 0, include_self = FALSE, sort_by = "weight")
```

Arguments

model	A netobject, TNA model object, mcm1 object, or a matrix of weights.
threshold	Numeric. Minimum weight to include an edge. Default: 0.
include_self	Logical. Whether to include self-loops. Default: FALSE.
sort_by	Character. Column to sort by: "weight" (descending), "from", "to", or NULL for no sorting. Default: "weight".

Details

This function converts the transition matrix into an edge list format, which is useful for visualization, analysis with igraph, or export to other network tools.

Value

For a single network: a data frame with one row per retained edge and columns:

from Source state name.

to Target state name.

weight Edge weight (transition probability).

For an `mcml` object: a named list of such data frames, with macro first and then one element per cluster.

See Also

[extract_transition_matrix](#) for the full matrix, [build_network](#) for network estimation.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
net <- build_network(seqs, method = "relative")
edges <- extract_edges(net, threshold = 0.05)
head(edges)
```

`extract_initial_probs` *Extract Initial Probabilities from Model*

Description

Extract the initial state probability vector from a TNA model object.

Usage

```
extract_initial_probs(model)

## S3 method for class 'nest_initial_probs'
summary(object, ...)
```

Arguments

<code>model</code>	A <code>netobject</code> , TNA model object, <code>mcml</code> object, or a list containing an initial element.
<code>object</code>	For the <code>summary()</code> method: an object of class <code>nest_initial_probs</code> .
<code>...</code>	In <code>summary.nest_initial_probs()</code> : Additional arguments (ignored).

Details

Initial probabilities represent the probability of starting a sequence in each state. If the model doesn't have explicit initial probabilities, this function falls back to a uniform distribution over the states of the transition matrix and warns.

Value

For a single network: a named numeric vector of initial state probabilities summing to 1, of class `c("nest_initial_probs", "numeric")`. The class stamp only adds a `summary()` method returning a tidy state/prob data frame.

For an `mcml` object: a named list of such vectors, with macro first and then one element per cluster (taken from each layer's `$inits`, unstamped).

In `summary.nest_initial_probs()`: A tidy data frame with columns `state` and `prob`, sorted by decreasing probability.

See Also

[extract_transition_matrix](#) for extracting the transition matrix, [extract_edges](#) for extracting an edge list.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
net <- build_network(seqs, method = "relative")
init_probs <- extract_initial_probs(net)
print(init_probs)
```

extract_pathways	<i>Cut an Event Log into Pathways</i>
------------------	---------------------------------------

Description

Splits a long event log into pathways and returns them as a tidy data frame, one row per pathway, ready to tabulate against an outcome or to feed to a sequence model.

Usage

```
extract_pathways(
  data,
  action,
  group,
  order = NULL,
  type = c("unit", "segments", "anchored"),
  anchor = NULL,
  terminal = NULL,
  resolve = NULL,
  sep = " -> "
)
```

Arguments

data	A long-format data.frame, one row per event, already in chronological order within each group (or with order supplied).
action	Name of the column holding the state / event label.
group	Character vector of column names defining the unit a pathway may not cross – typically the actor, or actor and session.
order	Optional column giving the within-group event order. When NULL (default) the existing row order is used.
type	The cut to make: "unit" (default) one pathway per group. With terminal, the group is truncated at its last terminal state so the pathway ends on an outcome rather than on whatever followed. "segments" consecutive, non-overlapping pathways: the group is cut after every terminal state, so each pathway ends on one. Every event belongs to exactly one pathway. "anchored" one pathway per occurrence of anchor, closing at the next terminal state. Pathways may overlap and events between a terminal and the next anchor belong to none.
anchor	State that opens a pathway when type = "anchored". Ignored by the other cuts. Default NULL.
terminal	States that close a pathway, the closing state included. Required for "segments" and "anchored"; optional for "unit", where it truncates.
resolve	Optional named list giving a resolution label to append as the pathway's final state. Each element is a character vector of states; a pathway takes the label of the first element whose states all occur in it, so order the list from most specific to least. Pathways matching nothing are labelled "Unresolved". The label becomes the last element of path and the value of closes, and the raw final state is kept in ends.
sep	Separator used when pasting the path. Default " -> ".

Value

A data.frame with one row per pathway: the group columns, pathway (an integer id within group), path (the states pasted with sep), length, opens (first state) and closes (last state). With resolve, closes holds the resolution label instead and a further ends column carries the raw final state. Pathways are returned in the order they occur.

See Also

[sequence_compare](#), [outcome_model](#)

Examples

```
log <- data.frame(
  id = c(1, 1, 1, 1, 1, 1, 2, 2, 2),
  act = c("Try", "Wrong", "Hint", "Retry", "Right", "Praise",
```

```

      "Try", "Wrong", "Retry"),
    stringsAsFactors = FALSE
  )
  # Each failure and its consequence:
  extract_pathways(log, action = "act", group = "id", type = "anchored",
    anchor = "Wrong", terminal = c("Right", "Wrong"))

  # Each id as one episode:
  extract_pathways(log, action = "act", group = "id")

```

```
extract_transition_matrix
```

Extract Transition Matrix from Model

Description

Extract the transition probability matrix from a TNA model object.

Usage

```
extract_transition_matrix(model, type = c("raw", "scaled"))
```

```
## S3 method for class 'nest_transition_matrix'
summary(object, ...)
```

Arguments

model	A netobject, TNA model object, mcml object, a list containing a weights element, or a bare weight matrix.
type	Character. Type of matrix to return: "raw" The raw weight matrix as stored in the model. "scaled" Row-normalized to ensure rows sum to 1. Default: "raw".
object	For the summary() method: an object of class nest_transition_matrix.
...	In summary.nest_transition_matrix(): Additional arguments (ignored).

Details

TNA models store transition weights in different locations depending on the model type. This function handles the extraction automatically.

For "scaled" type, each row is divided by its sum to create valid transition probabilities. This is useful when the original weights don't sum to 1.

Value

For a single network: a square numeric matrix with row and column names as state names, of class `c("nest_transition_matrix", "matrix", "array")`. It behaves as an ordinary matrix; the class stamp only adds a `summary()` method returning a tidy `from/to/weight` data frame.

For an `mcml` object: a named list of such matrices, with `macro` first and then one element per cluster.

In `summary.nest_transition_matrix()`: A tidy data frame with columns `from`, `to`, `weight`, with one row per non-zero entry.

See Also

[extract_initial_probs](#) for extracting initial probabilities, [extract_edges](#) for extracting an edge list.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
net <- build_network(seqs, method = "relative")
trans_mat <- extract_transition_matrix(net)
print(trans_mat)
```

frequencies

Build a Transition Frequency Matrix

Description

Convert long or wide format sequence data into a transition frequency matrix. Counts how many times each transition from `state_i` to `state_j` occurs across all sequences.

Usage

```
frequencies(
  data,
  action = "Action",
  id = NULL,
  time = "Time",
  cols = NULL,
  format = c("auto", "long", "wide")
)

## S3 method for class 'nest_transition_counts'
summary(object, ...)
```

Arguments

<code>data</code>	Data frame containing sequence data in long or wide format.
<code>action</code>	Character. Name of the column containing actions/states (for long format). Default: "Action".
<code>id</code>	Character vector. Name(s) of the column(s) identifying sequences. For long format, each unique combination of ID values defines a sequence. For wide format, used to exclude non-state columns. Default: NULL.
<code>time</code>	Character. Name of the time column used to order actions within sequences (for long format). Default: "Time".
<code>cols</code>	Character vector. Names of columns containing states (for wide format). If NULL, all non-ID columns are used. Default: NULL.
<code>format</code>	Character. Format of input data: "auto" (detect automatically), "long", or "wide". Default: "auto".
<code>object</code>	For the <code>summary()</code> method: an object of class <code>nest_transition_counts</code> .
<code>...</code>	In <code>summary.nest_transition_counts()</code> : Additional arguments (ignored).

Details

For **long format** data, each row is a single action/event. Sequences are defined by the `id` column(s), and actions are ordered by the `time` column within each sequence. Consecutive actions within a sequence form transition pairs.

For **wide format** data, each row is a sequence and columns represent consecutive time points. Transitions are counted across consecutive columns, skipping any NA values.

Value

A square integer matrix of transition frequencies, of class `c("nest_transition_counts", "matrix", "array")`, where `mat[i, j]` is the number of times state `i` was followed by state `j`. Row and column names are the sorted unique states. It behaves as an ordinary matrix and can be passed directly to `tna::tna()`; the class stamp only adds a `summary()` method, which returns the same counts as a tidy `from/to/count` data frame.

In `summary.nest_transition_counts()`: A tidy data frame with columns `from`, `to`, `count`, with one row per non-zero transition.

See Also

[convert_sequence_format](#) for converting to other representations (frequency counts, one-hot, edge lists).

Examples

```
# Wide format
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
freq <- frequencies(seqs, format = "wide")

# Long format
```

```
long <- data.frame(
  Actor = rep(1:2, each = 3), Time = rep(1:3, 2),
  Action = c("A", "B", "C", "B", "A", "C")
)
freq <- frequencies(long, action = "Action", id = "Actor")
```

get_estimator	<i>Retrieve a Registered Estimator</i>
---------------	--

Description

Retrieve a registered network estimator by name.

Usage

```
get_estimator(name)
```

Arguments

name	Character. Name of the estimator to retrieve.
------	---

Value

A list with elements fn, description, directed.

See Also

[register_estimator](#), [list_estimators](#)

Examples

```
est <- get_estimator("relative")
```

group_regulation_long	<i>Group Regulation in Collaborative Learning (Long Format)</i>
-----------------------	---

Description

Students' regulation strategies during collaborative learning, in long format. Contains 27,533 time-stamped action records from multiple students working in groups across two courses.

Usage

```
group_regulation_long
```

Format

A data frame with 27,533 rows and 6 columns:

Actor Integer. Student identifier.

Achiever Character. Achievement level: "High" or "Low".

Group Numeric. Collaboration group identifier.

Course Character. Course identifier ("A", "B", or "C").

Time POSIXct. Timestamp of the action.

Action Character. Regulation action (e.g., cohesion, consensus, discuss, synthesis).

Source

Synthetically generated from the group_regulation dataset in the **tna** package.

See Also

[learning_activities](#), [srl_strategies](#)

Examples

```
net <- build_network(group_regulation_long,
                     method = "relative",
                     actor = "Actor", action = "Action", time = "Time")
net
```

hypergraph_centrality *Hypergraph eigenvector centralities*

Description

Computes one or more eigenvector-style centralities on a [net_hypergraph](#): *clique-motif* (CEC), *Z-eigenvector* (ZEC), and *H-eigenvector* (HEC). Each variant captures influence differently - CEC flattens group structure via clique expansion, while ZEC and HEC propagate through the higher-order groups directly.

Usage

```
hypergraph_centrality(
  hg,
  type = c("clique", "Z", "H"),
  max_iter = 1000L,
  tol = 1e-08,
  normalize = TRUE
)
```

Arguments

hg	A net_hypergraph (from <code>build_hypergraph()</code> or <code>bipartite_groups()</code>).
type	Character vector, any subset of c("clique", "Z", "H"). Default computes all three.
max_iter	Maximum number of power-iteration steps. Default 1000.
tol	Convergence tolerance on the L1 change between successive iterates. Default 1e-8.
normalize	Logical. If TRUE (default), each returned centrality vector is L2-normalized to unit norm (compatible with <code>igraph::eigen_centrality()</code> 's scale for type "clique"). If FALSE, the vector is rescaled so its largest absolute entry is 1.

Details

Clique-motif eigenvector centrality (CEC): forms the clique-expanded pairwise graph W where $W_{ij} = |\{e : i, j \in e\}|$ and returns the leading eigenvector of W . Equivalent to running `igraph::eigen_centrality()` on `clique_expansion()` output.

Z-eigenvector centrality (ZEC): solves the linear eigen-equation on the hyperedge tensor,

$$\lambda x_i = \sum_{e \ni i} \prod_{j \in e, j \neq i} x_j,$$

via power iteration. Works for hypergraphs with mixed edge sizes.

H-eigenvector centrality (HEC): solves the power-k-1 eigen-equation,

$$\lambda x_i^{k-1} = \sum_{e \ni i} \prod_{j \in e, j \neq i} x_j.$$

For uniform hypergraphs (all hyperedges of size k), this is equivalent to normalizing the ZEC update by the geometric-mean exponent $1/(k-1)$. For mixed sizes, the effective exponent is taken from the largest hyperedge; expect slightly different rankings from ZEC in the mixed case.

Value

A named list, one component per requested type, in the order given by `type`. Each component is a numeric vector of length `hg$n_nodes` named by `hg$nodes`. A hypergraph with no hyperedges yields all-zero vectors.

Note

The "clique" (CEC) variant is validated against `igraph::eigen_centrality` (cosine ~ 1). The "Z" and "H" variants are **(experimental)** - validated only against a clean-room list-based tensor power iteration (same operator, different loop structure); no R package exposes tensor eigenvectors as a primitive for independent comparison.

References

Benson, A. R. (2019). Three hypergraph eigenvector centralities. *SIAM Journal on Mathematics of Data Science* 1(2), 293-312. arXiv:1807.09644.

See Also

[build_hypergraph\(\)](#), [clique_expansion\(\)](#), [hypergraph_measures\(\)](#).

Examples

```
df <- data.frame(
  player = c("A", "B", "C", "A", "B", "D", "C", "D", "E"),
  session = c("S1", "S1", "S1", "S2", "S2", "S3", "S3", "S3", "S3")
)
hg <- bipartite_groups(df, "player", "session")
cent <- hypergraph_centrality(hg)
# Compare rankings across the three variants
do.call(cbind, cent)
```

hypergraph_cluster	<i>Spectral clustering of hypergraph vertices</i>
--------------------	---

Description

Partitions the nodes of a hypergraph into k clusters with the Laplacian-eigenmap + k -means algorithm of Hayashi et al. (2020, "RDC-Spec"): the eigenvectors of the k smallest eigenvalues of the normalized hypergraph Laplacian are row-normalized to unit length and clustered with k -means. With `type = "random_walk"` and a weighted incidence (e.g. from [bipartite_groups\(\)](#) with `weight =`), the edge-dependent vertex weights genuinely change the partition - with edge-independent weights the walk collapses to a graph random walk (Chitra & Raphael 2019).

Usage

```
hypergraph_cluster(
  hg,
  k,
  type = c("zhou", "random_walk"),
  edge_weights = NULL,
  nstart = 25L,
  seed = NULL
)

## S3 method for class 'net_hypergraph_cluster'
print(x, ...)

## S3 method for class 'net_hypergraph_cluster'
summary(object, ...)

## S3 method for class 'net_hypergraph_cluster'
as.data.frame(x, ...)

## S3 method for class 'net_hypergraph_cluster'
plot(x, what = c("both", "spectrum", "embedding"), n_values = NULL, ...)
```

Arguments

hg	A connected net_hypergraph.
k	Integer number of clusters, $2 \leq k \leq n_nodes - 1$.
type, edge_weights	Passed to hypergraph_laplacian() .
nstart	Integer. k-means random restarts (default 25).
seed	Optional integer seed for the k-means initialization.
x	For the print(), as.data.frame() and plot() methods: an object of class net_hypergraph_cluster.
...	In as.data.frame.net_hypergraph_cluster(), plot.net_hypergraph_cluster(), print.net_hypergraph_cluster() and summary.net_hypergraph_cluster(): Additional arguments (ignored).
object	For the summary() method: an object of class net_hypergraph_cluster.
what	Character. "both" (default), "spectrum", or "embedding".
n_values	Integer. How many smallest eigenvalues to show in the spectrum panel (default: $\min(3 * k, n_nodes)$).

Details

k-means is stochastic: nstart restarts are used and a seed fixes the result. Report stability across seeds for consequential results.

Value

An object of class net_hypergraph_cluster: a list with \$clusters (data.frame, one row per node: node, cluster - labels "Cluster 1", "Cluster 2", ... ordered by first appearance), \$embedding (node x k row-normalized spectral embedding used by k-means, dims dim1..dimk), \$k, \$type, \$eigenvalues (full Laplacian spectrum, increasing), \$eigengap (gap after the k-th eigenvalue), \$sizes (data.frame cluster/size), \$pi (named stationary distribution), \$n_nodes, \$n_hyperedges and \$params (the edge_weights used, nstart, seed, tot_withinss). Has print, summary, plot and as.data.frame methods; as.data.frame() returns one row per node with node, cluster, the stationary probability pi, and the embedding coordinates.

In print.net_hypergraph_cluster(): The input object, invisibly.

In summary.net_hypergraph_cluster(): A data.frame, one row per cluster: cluster, size, share.

In as.data.frame.net_hypergraph_cluster(): The tidy assignment table: one row per node, columns node, cluster, pi (stationary probability of the node under the Laplacian's random walk) and the spectral-embedding coordinates dim1..dimk.

In plot.net_hypergraph_cluster(): For "spectrum"/"embedding", the ggplot object. For "both", the arranged gtable when gridExtra is installed (drawn on the current device), otherwise the two panels are drawn via grid viewports and the list of the two ggplots is returned invisibly.

Methods

- `plot.net_hypergraph_cluster()`: Two diagnostic panels. "spectrum": scree plot of the Laplacian spectrum with the `k` used for clustering marked - the eigengap after `k` supports (or questions) the choice of `k`. "embedding": the nodes in the first two spectral-embedding dimensions, labelled, coloured and shaped by cluster, sized by stationary probability - the geometry `k`-means actually clustered. "both" (default) arranges the two side by side (via `gridExtra` when available, base grid viewports otherwise).

References

- Hayashi, K., Aksoy, S. G., Park, C. H., & Park, H. (2020). Hypergraph random walks, Laplacians, and clustering. *CIKM 2020*, 495-504. doi:[10.1145/3340531.3412034](https://doi.org/10.1145/3340531.3412034)
- Chitra, U., & Raphael, B. J. (2019). Random walks on hypergraphs with edge-dependent vertex weights. *ICML 2019*.

Examples

```
events <- data.frame(
  person = c("a", "b", "c", "a", "b", "c", "d", "e", "f",
             "d", "e", "f", "c", "d"),
  meeting = c("m1", "m1", "m1", "m2", "m2", "m2", "m3", "m3", "m3",
              "m4", "m4", "m4", "m5", "m5")
)
hg <- bipartite_groups(events, player = "person", group = "meeting")
cl <- hypergraph_cluster(hg, k = 2, seed = 1)
cl
as.data.frame(cl)
```

hypergraph_laplacian *Normalized hypergraph Laplacian*

Description

Computes the normalized Laplacian of a hypergraph, either the classic Zhou-Huang-Scholkopf form on the binary incidence pattern (`type = "zhou"`) or the random-walk form with edge-dependent vertex weights (`type = "random_walk"`), in which the weighted incidence cells (e.g. the summed weights produced by `bipartite_groups()`) determine where a random walker lands inside a hyperedge, and the resulting non-reversible walk is symmetrized through its stationary distribution (Chung 2005). Both Laplacians are symmetric positive semi-definite with eigenvalues in $[0, 2]$; for a binary incidence with unit hyperedge weights they coincide.

Usage

```
hypergraph_laplacian(hg, type = c("zhou", "random_walk"), edge_weights = NULL)
```

Arguments

hg	A net_hypergraph from <code>build_hypergraph()</code> or <code>bipartite_groups()</code> . Must be connected and have at least one hyperedge.
type	Character. "zhou" (default) for the Zhou et al. (2006) normalized Laplacian on the binary incidence pattern, or "random_walk" for the Hayashi et al. (2020) EDVW random-walk Laplacian on the weighted incidence.
edge_weights	Numeric vector of positive hyperedge weights (length <code>hg\$n_hyperedges</code>), or NULL for the type-specific default: unit weights for "zhou"; for "random_walk" the Hayashi et al. heuristic - the population standard deviation of each hyperedge's (non-zero) vertex weights plus one - which reduces to unit weights on a binary incidence.

Value

A symmetric `n_nodes` x `n_nodes` numeric matrix (node names as dimnames) with attributes `type` (the Laplacian type), `pi` (named stationary distribution of the underlying random walk) and `edge_weights` (the hyperedge weights actually used).

References

- Zhou, D., Huang, J., & Scholkopf, B. (2006). Learning with hypergraphs: Clustering, classification, and embedding. *NeurIPS 19*.
- Hayashi, K., Aksoy, S. G., Park, C. H., & Park, H. (2020). Hypergraph random walks, Laplacians, and clustering. *CIKM 2020*, 495-504. doi:10.1145/3340531.3412034
- Chung, F. (2005). Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1), 1-19.

Examples

```
events <- data.frame(
  person = c("a", "b", "c", "a", "b", "d", "c", "d", "e", "e", "a"),
  meeting = c("m1", "m1", "m1", "m2", "m2", "m2", "m3", "m3", "m3",
              "m4", "m4"),
  hours = c(2, 1, 1, 3, 2, 1, 2, 2, 4, 1, 1)
)
hg <- bipartite_groups(events, player = "person", group = "meeting",
                      weight = "hours")
L <- hypergraph_laplacian(hg, type = "random_walk")
range(eigen(L, symmetric = TRUE, only.values = TRUE)$values)
```

hypergraph_measures *Structural measures for a hypergraph*

Description

Computes a comprehensive structural-statistics suite for a `net_hypergraph`: node-level, hyperedge-level, and global measures. All measures are derived in a few BLAS calls on the incidence matrix.

Usage

```
hypergraph_measures(hg)

## S3 method for class 'hypergraph_measures'
print(x, ...)
```

Arguments

hg	A net_hypergraph (from build_hypergraph() or bipartite_groups()).
x	For the print() method: an object of class hypergraph_measures.
...	In print.hypergraph_measures(): Additional arguments (ignored).

Details

All measures are computed via standard matrix operations on the binary incidence $B = (b_{ij})$ where $b_{ij} = 1$ iff node i is in hyperedge j :

- `hyperdegree` = `rowSums(B)`, `edge_sizes` = `colSums(B)`
- `co_degree` = `tcrossprod(B)` (with zero diagonal)
- `edge_pairwise_overlap` = `crossprod(B)` (with zero diagonal)
- `overlap_coefficient[i, j]` = `overlap[i, j] / min(edge_sizes[i], edge_sizes[j])`
- `jaccard[i, j]` = `overlap[i, j] / (edge_sizes[i] + edge_sizes[j] - overlap[i, j])`

Empty hypergraph (`n_hyperedges == 0`) returns trivial zeros and empty matrices.

Value

An object of class `hypergraph_measures` (a named list) with components:

Node-level (length `n_nodes`)

`hyperdegree` Number of hyperedges containing each node.

`node_strength` Total participation: for node i , $\sum_{e \ni i} |e|$. A node in many large hyperedges has high strength.

`max_edge_size` Size of the largest hyperedge containing each node.

`co_degree` `n_nodes` x `n_nodes` matrix: `co_degree[i, j]` = $|\{e : i, j \in e\}|$ (number of hyperedges co-containing nodes i and j). Diagonal is zero.

Hyperedge-level (length `n_hyperedges` or `m` x `m`)

`edge_sizes` Hyperedge sizes $|e|$.

`edge_pairwise_overlap` `m` x `m` matrix: $|e_i \cap e_j|$. Diagonal is zero.

`overlap_coefficient` `m` x `m`: $|e_i \cap e_j| / \min(|e_i|, |e_j|)$. Measures how much the smaller hyperedge is contained in the larger.

`jaccard` `m` x `m`: symmetric overlap index $|e_i \cap e_j| / |e_i \cup e_j|$.

Global (scalars)

`density` For k -uniform hypergraphs: $m / \text{choose}(n, k)$. For mixed sizes: $\text{sum}(|e|) / (n * m)$ (mean fraction of nodes per hyperedge).

`avg_edge_size` Mean of `edge_sizes`.
`size_distribution` Tabulation of hyperedge sizes (passed through from `hg`).
`intersection_profile` Distribution of pairwise hyperedge intersection sizes - useful for spotting whether hyperedges overlap mostly trivially or share substantial cores (Do et al. 2020).
`pairwise_participation` Fraction of node pairs co-appearing in at least one hyperedge.
`n_nodes, n_hyperedges` Convenience scalars.

In `print.hypergraph_measures()`: The input `x` invisibly.

References

Lee, G., Choe, M., & Shin, K. (2021). How do hyperedges overlap in real-world hypergraphs? Patterns, measures, and generators. *Proceedings of the Web Conference 2021*, 3396-3407.
 Do, M. T., Yoon, S., Hooi, B., & Shin, K. (2020). Structural patterns and generative models of real-world hypergraphs. arXiv:2006.07060.

See Also

[build_hypergraph\(\)](#), [bipartite_groups\(\)](#), [clique_expansion\(\)](#).

Examples

```

df <- data.frame(
  player = c("A", "B", "C", "A", "B", "D", "C", "D"),
  session = c("S1", "S1", "S1", "S2", "S2", "S3", "S3", "S3")
)
hg <- bipartite_groups(df, "player", "session")
m <- hypergraph_measures(hg)
print(m)
m$hyperdegree      # how many sessions each player joined
m$co_degree        # pairwise co-membership counts
m$jaccard           # symmetric overlap between sessions
  
```

hypergraph_transduction

Transductive label spreading on a hypergraph

Description

Semi-supervised classification of hypergraph nodes by the regularization framework of Zhou et al. (2006): given labels for a subset of nodes, the scores $F = (I - \xi) * (I - \xi * S)^{-1} Y$ spread the labels over the hypergraph, where $S = I - L$ is the normalized similarity operator of the chosen Laplacian and Y is the label indicator matrix. Each node is assigned the class with the highest score. This is the non-neural ancestor of hypergraph-attention text classifiers: with documents as hyperedges over words (or vice versa) it classifies unlabeled nodes from a handful of labeled ones.

Usage

```

hypergraph_transduction(
  hg,
  labels,
  xi = 0.99,
  type = c("zhou", "random_walk"),
  edge_weights = NULL
)

## S3 method for class 'net_hypergraph_transduction'
print(x, ...)

## S3 method for class 'net_hypergraph_transduction'
summary(object, ...)

## S3 method for class 'net_hypergraph_transduction'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  what = c("predictions", "scores"),
  ...
)

## S3 method for class 'net_hypergraph_transduction'
plot(x, ...)

```

Arguments

hg	A connected net_hypergraph.
labels	Node labels. Either a named vector (names = node names, values = class labels) covering a subset of nodes, or a full-length vector aligned with hg\$nodes with NA for unlabeled nodes. At least two distinct classes must be labeled.
xi	Numeric in (0, 1). Spreading coefficient (default 0.99); larger values weight the hypergraph structure more relative to the initial labels.
type, edge_weights	Passed to hypergraph_laplacian() .
x	For the print(), as.data.frame() and plot() methods: an object of class net_hypergraph_transduction.
...	In as.data.frame.net_hypergraph_transduction(), plot.net_hypergraph_transduction(), print.net_hypergraph_transduction() and summary.net_hypergraph_transduction(): Additional arguments (ignored).
object	For the summary() method: an object of class net_hypergraph_transduction.
row.names	NULL (default) or a character vector of row names for the returned data frame.
optional	Ignored; present so the method matches the signature of the as.data.frame() generic.

what Character. "predictions" (default) for the one-row-per-node table, "scores" for the tidy long score table (one row per node x class: node, class, score).

Value

An object of class `net_hypergraph_transduction`: a list with `$predictions` (data.frame, one row per node: node, label (given, NA if unlabeled), predicted, score (winning class score), margin (winning minus runner-up score)), `$classes`, `$scores` (node x class score matrix), `$xi`, `$type`, `$n_labeled`, `$n_nodes` and `$params` (the edge_weights used). Has `print`, `summary`, `plot` and `as.data.frame` methods; `as.data.frame(x, what = "scores")` returns the tidy long score table.

In `print.net_hypergraph_transduction()`: The input object, invisibly.

In `summary.net_hypergraph_transduction()`: A data.frame, one row per class: class, n_labeled, n_predicted, mean_margin (mean winning margin among the nodes predicted into the class).

In `as.data.frame.net_hypergraph_transduction()`: A data.frame selected by what: for "predictions", one row per node with columns node, label (the given label, NA if unlabeled), predicted, score and margin; for "scores", one row per node x class with columns node, class and score.

In `plot.net_hypergraph_transduction()`: A ggplot object (the score heatmap), returned visibly so that `plot(x)` draws it.

Methods

- `plot.net_hypergraph_transduction()`: Heatmap of the full node-by-class score matrix: rows are nodes (grouped by predicted class), columns are classes, tile shading and printed values are the spreading scores. Seed nodes (given labels) carry a black tile border, and each node's winning class is marked with a dot, so agreement between seeds, scores, and decisions is visible in one panel. Rows whose winning and runner-up scores are close (small margin) are the assignments to distrust.

References

Zhou, D., Huang, J., & Scholkopf, B. (2006). Learning with hypergraphs: Clustering, classification, and embedding. *NeurIPS 19*.

Examples

```
events <- data.frame(
  person = c("a", "b", "c", "a", "b", "c", "d", "e", "f",
            "d", "e", "f", "c", "d"),
  meeting = c("m1", "m1", "m1", "m2", "m2", "m2", "m3", "m3", "m3",
            "m4", "m4", "m4", "m5", "m5")
)
hg <- bipartite_groups(events, player = "person", group = "meeting")
tr <- hypergraph_transduction(hg, labels = c(a = "x", d = "y"))
tr
as.data.frame(tr)
```

Description

The item table behind `build_mcml_pc`: one row per node, reporting how strongly the item connects to its own cluster, the weight it carries into that cluster's composite, and whether it connects more strongly to some other cluster.

Reading the table is `item_loadings(fit)` - never a reach into the fit's internals. The name says *item*: these are network loadings of items on their own cluster, not factor loadings.

Usage

```
item_loadings(x, ...)

## S3 method for class 'mcml_pc'
item_loadings(x, misfit = NULL, ...)
```

Arguments

<code>x</code>	An <code>mcml_pc</code> object from <code>build_mcml_pc</code> .
<code>...</code>	Ignored.
<code>misfit</code>	Logical or NULL. NULL (default) returns every item; TRUE returns only the items whose strongest cross-cluster connection exceeds their own-cluster loading; FALSE returns only the items that fit where they were assigned.

Value

A data frame with one row per node (one row per misfitting or fitting node when `misfit` is set) and columns `node`, `cluster`, `loading` (signed mean connection to its own cluster), `weight` (its composite weight), `sign` (+1, or -1 for a reverse-keyed item), `max_cross` (strongest connection to any other cluster), `cross_cluster` (which cluster that is), and `misfit` (logical).

See Also

`build_mcml_pc` to create the fit, `loading_stability` for the weights' sampling uncertainty, `composites` for the scores these weights produce.

Examples

```
set.seed(1)
f <- stats::rnorm(200)
g <- stats::rnorm(200)
df <- data.frame(a1 = f + stats::rnorm(200), a2 = f + stats::rnorm(200),
                 a3 = f + stats::rnorm(200), b1 = g + stats::rnorm(200),
                 b2 = g + stats::rnorm(200), b3 = g + stats::rnorm(200))
clusters <- list(A = c("a1", "a2", "a3"), B = c("b1", "b2", "b3"))
```

```
fit <- build_mcmc1_pc(df, clusters, aggregation = "loadings",
                     method = "cor")
item_loadings(fit)
item_loadings(fit, misfit = TRUE)
```

learning_activities	<i>Online Learning Activity Indicators</i>
---------------------	--

Description

Simulated binary time-series data for 200 students across 30 time points. At each time point, one or more learning activities may be active (1) or inactive (0). Activities: Reading, Video, Forum, Quiz, Coding, Review. Includes temporal persistence (activities tend to continue across adjacent time points).

Usage

```
learning_activities
```

Format

A data frame with 6,000 rows and 7 columns:

student Integer. Student identifier (1–200).

Reading Integer (0/1). Reading activity indicator.

Video Integer (0/1). Video watching indicator.

Forum Integer (0/1). Discussion forum indicator.

Quiz Integer (0/1). Quiz/assessment indicator.

Coding Integer (0/1). Coding practice indicator.

Review Integer (0/1). Review/revision indicator.

Examples

```
head(learning_activities)
```

list_estimators	<i>List All Registered Estimators</i>
-----------------	---------------------------------------

Description

Return a data frame summarising all registered network estimators.

Usage

```
list_estimators()
```

Value

A data frame with columns name, description, directed.

See Also

[register_estimator](#), [get_estimator](#)

Examples

```
list_estimators()
```

loading_stability	<i>Composite-Weight Stability Under Case Resampling</i>
-------------------	---

Description

Experimental. Bootstraps the item weights of a [build_mcml_pc](#) fit: rows of the raw data are resampled, the node-level network is re-estimated each time, and the connectivity-based composite weights are recomputed. Wide intervals mean the weighting (and therefore the "loadings" macro network) should not be over-interpreted.

Usage

```
loading_stability(x, iter = 200L, ci_level = 0.05, seed = NULL)

## S3 method for class 'pc_loading_stability'
print(x, digits = 3, ...)

## S3 method for class 'pc_loading_stability'
plot(x, ...)
```

Arguments

<code>x</code>	An <code>mcml_pc</code> object that carries raw data. For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>pc_loading_stability</code> .
<code>iter</code>	Integer. Bootstrap replicates (default 200; node-level re-estimation makes this heavier than a plain bootstrap).
<code>ci_level</code>	Numeric. Significance level for percentile CIs (default 0.05).
<code>seed</code>	Integer or NULL. RNG seed.
<code>digits</code>	Number of digits to display (default 3).
<code>...</code>	In <code>plot.pc_loading_stability()</code> and <code>print.pc_loading_stability()</code> : Additional arguments (ignored).

Value

An object of class `"pc_loading_stability"`: a list with summary (tidy data frame: `node`, `cluster`, `weight`, `boot_mean`, `boot_sd`, `ci_lower`, `ci_upper`, `sign_flips` - the proportion of replicates in which the item's sign differed from the observed one), `boot_weights` (`iter` x `n_nodes` matrix), `iter`, and `ci_level`. Has `print` and `plot` methods.

In `print.pc_loading_stability()`: `x`, invisibly.

In `plot.pc_loading_stability()`: A `ggplot` object.

Methods

- `plot.pc_loading_stability()`: Signed composite weights with bootstrap percentile intervals, faceted by cluster.

Examples

```
set.seed(1)
f <- stats::rnorm(100)
g <- stats::rnorm(100)
df <- data.frame(a1 = f + stats::rnorm(100), a2 = f + stats::rnorm(100),
                 a3 = f + stats::rnorm(100), b1 = g + stats::rnorm(100),
                 b2 = g + stats::rnorm(100), b3 = g + stats::rnorm(100))
cl <- list(A = c("a1", "a2", "a3"), B = c("b1", "b2", "b3"))
fit <- build_mcml_pc(df, cl, aggregation = "loadings",
                    method = "cor")
stability <- loading_stability(fit, iter = 50, seed = 1)
stability
```

long-data	<i>Human-AI Vibe Coding Interaction Data (Long Format)</i>
-----------	--

Description

Coded interaction sequences from 429 human-AI pair programming sessions across 34 projects, in long format.

Usage

human_long

ai_long

Format

Data frames in long format with 9 columns:

message_id Integer. Turn index.

project Character. Project identifier (Project_1 .. Project_34).

session_id Character. Unique session hash.

timestamp Integer. Unix timestamp for ordering.

session_date Character. Date of the session (YYYY-MM-DD).

code Character. Interaction code. The two data frames use disjoint code sets – 9 labels in human_long, 8 in ai_long, 17 in total.

cluster Character. High-level cluster. human_long uses Directive, Evaluative and Metacognitive; ai_long uses Action, Communication and Repair.

code_order Integer. Order of the code within the session.

order_in_session Integer. Absolute turn order within the session.

human_long	Human turns only, 10,796 rows
ai_long	AI turns only, 8,551 rows

An object of class `data.frame` with 10796 rows and 9 columns.

An object of class `data.frame` with 8551 rows and 9 columns.

Source

Saqr, M. (2026). Human-AI vibe coding interaction study. <https://saqr.me/blog/2026/human-ai-interaction-cograp>

Examples

```
net <- build_network(human_long, method = "tna",
                    action = "code", actor = "session_id",
                    time = "timestamp")

net
```

long_to_wide

*Convert Long Format to Wide Sequences***Description**

Convert sequence data from long format (one row per action) to wide format (one row per sequence, columns as time points).

Usage

```
long_to_wide(
  data,
  id_col = "Actor",
  time_col = "Time",
  action_col = "Action",
  time_prefix = "V",
  fill_na = TRUE
)
```

Arguments

data	Data frame in long format.
id_col	Character. Name of the column identifying sequences. Default: "Actor".
time_col	Character. Name of the column identifying time points. Default: "Time".
action_col	Character. Name of the column containing actions/states. Default: "Action".
time_prefix	Character. Prefix for time point columns in output. Default: "V".
fill_na	Logical. Whether to fill missing time points with NA. Default: TRUE.

Details

Converts long format data (one row per action) to the wide format expected by [build_network](#), `tna::tna()` and related functions.

If `time_col` contains non-integer values (e.g., timestamps), the function will use the ordering within each sequence to create time indices.

Value

A data frame in wide format, one row per sequence: the `id_col` column followed by the time point columns `V1`, `V2`, ... (named with `time_prefix`) holding the action at each time point. With `fill_na = TRUE` short sequences are padded with NA so every row shares the same columns; with `fill_na = FALSE` only the time points present in every sequence are kept.

See Also

[wide_to_long](#) for the reverse conversion, [prepare_for_tna](#) for preparing data for TNA analysis.

Examples

```
long_data <- data.frame(
  Actor = rep(1:3, each = 4),
  Time = rep(1:4, 3),
  Action = sample(c("A", "B", "C"), 12, replace = TRUE)
)
wide_data <- long_to_wide(long_data, id_col = "Actor")
head(wide_data)
```

macro_network	<i>Cluster-Level Network, With One Cluster Expanded</i>
---------------	---

Description

Returns the macro (cluster-level) network of an `mcml`, optionally with one or more clusters expanded back into their member states. Every other cluster stays collapsed to a single node, so the result is a network at mixed resolution: the cluster of interest in detail, its context in summary.

Usage

```
macro_network(x, expand = NULL, method = "relative", ...)
```

Arguments

- | | |
|--------|--|
| x | An <code>mcml</code> built from sequence data, or an <code>mcml_pc</code> from build_mcml_pc . A matrix-derived <code>mcml</code> carries no node-level data and cannot be expanded. |
| expand | Names of clusters to expand into their member states. NULL (default) collapses every cluster, reproducing the macro layer; "all" or TRUE expands every cluster, so each state is its own node. |
| method | Estimator passed to build_network . Default "relative" (row-normalised transitions). Not used for an <code>mcml_pc</code> . |
| ... | Further arguments passed to build_network . Not used for an <code>mcml_pc</code> . |

Value

For an `mcml_pc`: its cluster-level network, the `netobject` estimated by `build_mcml_pc`, unchanged. Its estimator is set when the fit is built, so `method` or `...` raise an error, and expand errors with class `nestimate_no_expand` (there are no sequences to re-count).

For an `mcml`: a `netobject` (also a `cograph_network`) whose nodes are the collapsed clusters plus the member states of any expanded cluster, with weights re-counted from the sequence data by `build_network`. `$node_groups` is a two-column data frame (`node`, `group`) mapping every node to its cluster, and the same labels are a factor in `$nodes$groups`, so the result plots grouped; an expanded cluster's states each map to that cluster, a collapsed cluster maps to itself. `$expanded` records the cluster names that were expanded (NULL when none were).

See Also

`build_mcml`, `as_tna`

Examples

```
seqs <- data.frame(
  t1 = c("A", "C", "A", "B"), t2 = c("B", "D", "C", "A"),
  t3 = c("C", "A", "D", "C"), stringsAsFactors = FALSE
)
mc <- build_mcml(seqs, clusters = list(G1 = c("A", "B"), G2 = c("C", "D")))
macro_network(mc)                # every cluster collapsed
macro_network(mc, expand = "G2") # G2 shown as C and D
```

magnitude_difference *Magnitude difference between the frequency and probability views*

Description

Quantifies, per edge, how much row-normalization moves a transition network between its two natural summaries: raw transition counts (frequency / FTNA, `build_network(method = "frequency")`) and row-conditional probabilities (TNA, `build_network(method = "relative")`). The two matrices rank edges differently - an edge that is large in counts can be modest in probability, and a rare-source edge can dominate its row in probability. The per-edge discrepancy on a common scale is the *magnitude difference*.

Usage

```
magnitude_difference(
  data,
  actor = "Actor",
  action = "Action",
  time = NULL,
  metric = c("abs_diff", "chord_dist", "atanh_diff", "geom_norm_diff", "cv_inflation"),
  scale = c("tna_range", "rank_minmax", "minmax", "none"),
  format = c("auto", "long", "wide")
)
```

```

)

## S3 method for class 'magnitude_difference'
print(x, ...)

## S3 method for class 'magnitude_difference'
plot(x, type = c("stacked", "circular"), min_show = 0.01, title = NULL, ...)

```

Arguments

<code>data</code>	Long- or wide-format event log (<code>data.frame</code>).
<code>actor, action, time</code>	Column names in <code>data</code> (long format). <code>time</code> may be <code>NULL</code> .
<code>metric</code>	Discrepancy metric. One of <code>"abs_diff"</code> (default; absolute difference, simplest and most robust), <code>"chord_dist"</code> (chord distance on the unit sphere), <code>"atanh_diff"</code> (Fisher z-style on a bounded scale), <code>"geom_norm_diff"</code> (geometric-mean normalized; amplifies small edges), <code>"cv_inflation"</code> (per-vector SD-standardized then absolute difference).
<code>scale</code>	How the two weight matrices are placed on a common scale before differencing. <code>"tna_range"</code> (default) rescales FTNA linearly into TNA's <code>[min, max]</code> range and leaves TNA untouched, so the difference is in TNA probability units. <code>"rank_minmax"</code> converts each matrix's values to ranks scaled to <code>[0, 1]</code> (ordinal). <code>"minmax"</code> scales each matrix's raw values to <code>[0, 1]</code> separately (asymmetric - TNA's max and FTNA's max map to the same value despite differing native ranges). <code>"none"</code> uses raw weights.
<code>format</code>	Input format passed through to <code>build_network()</code> ; <code>"auto"</code> (default) treats the data as wide when <code>action</code> is not a column.
<code>x</code>	For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>magnitude_difference</code> .
<code>...</code>	In <code>plot.magnitude_difference()</code> : Ignored. In <code>print.magnitude_difference()</code> : Passed to plotting helpers (ignored by <code>print</code>).
<code>type</code>	Plot style, <code>"stacked"</code> (default) or <code>"circular"</code> .
<code>min_show</code>	For <code>type = "circular"</code> , drop edges whose magnitude is below this fraction of the maximum.
<code>title</code>	Plot title. <code>NULL</code> generates one from the metric and scale.

Value

An object of class `"magnitude_difference"`: a list with `$edges` (per-edge `data.frame` with columns `from`, `to`, `ftna`, `tna`, `signed = tna - ftna`, and `value = the chosen metric`), `$metric`, `$scale`, `$weights_ftna`, `$weights_tna`, and `$states`.

In `print.magnitude_difference()`: `print` invisibly returns `x`.

In `plot.magnitude_difference()`: `plot` returns a `ggplot` object.

See Also

[build_network\(\)](#), [compare_model\(\)](#)

Examples

```
data(group_regulation_long, package = "Nestimate")
fit <- magnitude_difference(group_regulation_long,
                           actor = "Actor", action = "Action",
                           time = "Time")

print(fit)

# The polar portrait shows which edges row-normalization promotes.
plot(fit) # stacked polar portrait
plot(fit, type = "circular") # chord-style diagram
```

markov_order_test	<i>Test the Markov order of a sequential process</i>
-------------------	--

Description

Principled test of whether a categorical sequence is best described as a k -th order Markov chain. At each order $k = 1, \dots, \text{max_order}$, the function computes the classical likelihood-ratio statistic (G^2) for the conditional independence $s \perp x \mid w$, where w is the $(k - 1)$ -gram context, x is the extra (k -th-back) state added at order k , and s is the next state. Under H_0 (order- $(k - 1)$ is correct), s is independent of x given w .

The null distribution is obtained by an **exact within- w permutation test**: for each context w the successor labels are exchangeable under H_0 , so shuffling s within each w -group yields an exact reference distribution for G^2 . No plug-in MLE bias and no refitting per replicate. An asymptotic χ^2 p-value is reported alongside for reference.

Order selection is sequential: the order is raised while the test rejects, and stops at the first non-rejection. The reported `optimal_order` is therefore the highest k whose test - and every test below it - rejected at level α , i.e. one below the first non-rejection; it is $\emptyset$ when order 1 is already not rejected, and `max_order` when no test accepts.

Usage

```
markov_order_test(
  data,
  max_order = 3L,
  n_perm = 500L,
  alpha = 0.05,
  parallel = FALSE,
  n_cores = 2L,
  seed = NULL
)

## S3 method for class 'net_markov_order'
print(x, ...)

## S3 method for class 'net_markov_order_group'
```

```

print(x, ...)

## S3 method for class 'net_markov_order'
summary(object, ...)

## S3 method for class 'net_markov_order'
plot(x, panel = c("both", "ic", "permutation"), combined = TRUE, ...)

```

Arguments

<code>data</code>	A data.frame (wide format, one sequence per row), a list of character vectors (one per trajectory), a netobject or netobject_group carrying its \$data, or a prepare result (its sequence_data is used). NAs are treated as end of sequence.
<code>max_order</code>	Integer. Highest Markov order to test. Default 3.
<code>n_perm</code>	Integer. Number of within- <i>w</i> permutations per order. Default 500.
<code>alpha</code>	Numeric. Significance level for order selection. Default 0.05.
<code>parallel</code>	Logical. Use <code>parallel::mclapply</code> for permutations. Default FALSE (set TRUE only on Unix-like systems).
<code>n_cores</code>	Integer. Cores for parallel execution. Default 2.
<code>seed</code>	Optional integer seed for reproducibility.
<code>x</code>	For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_markov_order</code> or <code>net_markov_order_group</code> .
<code>...</code>	In <code>plot.net_markov_order()</code> , <code>print.net_markov_order()</code> and <code>summary.net_markov_order()</code> : Ignored. In <code>print.net_markov_order_group()</code> : Forwarded to <code>print.net_markov_order</code> for each element.
<code>object</code>	For the <code>summary()</code> method: an object of class <code>net_markov_order</code> .
<code>panel</code>	Which panel(s) to render: "both", "ic", or "permutation". Default "both".
<code>combined</code>	When <code>panel = "both"</code> and <code>combined = TRUE</code> (default), the two panels are drawn side-by-side. If gridExtra is installed they are arranged into a single drawable/saveable gtable (returned); otherwise base grid viewports draw both panels and a named list of the two ggplots is returned invisibly. When FALSE, returns that named list (ic, permutation) without drawing. Ignored when <code>panel != "both"</code> .

Value

An object of class `net_markov_order` with elements:

optimal_order Integer. Selected order via sequential permutation test.

bic_order Integer. Order minimising BIC (reported by `print/plot`; not used in the permutation selection).

aic_order Integer. Order minimising AIC (reported by `print/plot`; not used in the permutation selection).

test_table Tidy data.frame, one row per order tested with columns order, loglik, AIC, BIC, df, g2, p_permutation, p_asymptotic, significant. AIC/BIC are the information-criterion values used for the ic plot panel and to derive aic_order/bic_order; the order-0 row has NA for the test columns (df, g2, p_permutation, p_asymptotic, significant).

permutation_null List of numeric vectors (length max_order), one empirical null G^2 distribution per order.

logliks Named numeric vector of log-likelihoods per order (for AIC / BIC panel only, not used in the test).

layer_dofs Named integer vector of model degrees of freedom per order (free parameters added at each layer), used to compute the AIC / BIC columns.

transition_matrices List of fitted transition matrices.

states Character vector of observed state labels.

n_sequences, n_observations Data summary.

n_perm, alpha, max_order Call settings. max_order is the order actually tested, which is capped at length(longest sequence) - 1 with a message.

For a netobject_group the result is a "net_markov_order_group": a named list holding one net_markov_order per group.

In print.net_markov_order(): The input object, invisibly.

In print.net_markov_order_group(): x invisibly.

In summary.net_markov_order(): The tidy test_table data.frame - one row per order tested - carrying the selection context as attributes: optimal_order, bic_order, aic_order, alpha and n_perm.

In plot.net_markov_order(): A ggplot (single panel); for panel = "both", either a gridExtra ttable (when **gridExtra** is installed) or a named list of two ggplots (ic, permutation) drawn side-by-side and returned invisibly.

Plot panels

Two-panel professional visualization:

- Panel A: log-likelihood, AIC, BIC across tested orders with the selected order highlighted (both the permutation-selected order and the BIC-minimizing order are marked).
- Panel B: permutation null density per order with the observed G^2 as a vertical marker; colored by rejection at alpha.

Uses the Okabe-Ito colorblind-safe palette.

Examples

```
# Is one previous state enough to predict the next one?
res <- markov_order_test(as.data.frame(trjectories),
                        max_order = 2, n_perm = 99, seed = 1)

res
summary(res)
plot(res)
```

Description

Computes per-state stability metrics from a transition network: persistence (self-loop probability), stationary distribution, mean recurrence time, sojourn time, and mean accessibility to/from other states.

Usage

```
markov_stability(x, normalize = TRUE)

## S3 method for class 'net_markov_stability'
print(x, ...)

## S3 method for class 'net_markov_stability_group'
print(x, ...)

## S3 method for class 'net_markov_stability'
summary(object, ...)

## S3 method for class 'net_markov_stability'
plot(
  x,
  metrics = c("persistence", "stationary_prob", "return_time", "sojourn_time",
    "avg_time_to_others", "avg_time_from_others"),
  combined = TRUE,
  ...
)
```

Arguments

x	A netobject, cograph_network, tna object, row-stochastic numeric transition matrix, or a wide sequence data.frame (rows = actors, columns = time-steps). For the print() and plot() methods: an object of class net_markov_stability or net_markov_stability_group.
normalize	Logical. Normalize rows to sum to 1? Default TRUE.
...	Ignored. In plot.net_markov_stability(), print.net_markov_stability() and summary.net_markov_stability(): Ignored. In print.net_markov_stability_group(): Forwarded to print.net_markov_stability for each element.
object	For the summary() method: an object of class net_markov_stability.
metrics	Character vector. Which metrics to plot. Options: "persistence", "stationary_prob", "return_time", "sojourn_time", "avg_time_to_others", "avg_time_from_others". Default: all six.

combined When TRUE (default), all selected metrics are shown in one ggplot via `facet_wrap(~metric)`. When FALSE, returns a named list of single-panel ggplots, one per metric, so each can be printed, saved, or re-laid-out independently.

Details

Sojourn time is the expected consecutive time steps spent in a state before leaving: $1/(1 - P_{ii})$. States with persistence = 1 have `sojourn_time` = Inf.

avg_time_to_others: mean passage time from this state to all others; reflects how "sticky" or "isolated" the state is.

avg_time_from_others: mean passage time from all other states to this one; reflects accessibility (attractor strength).

Value

An object of class "net_markov_stability" with:

stability Data frame with one row per state and columns: `state`, `persistence` (P_{ii}), `stationary_prob` (π_i), `return_time` ($1/\pi_i$), `sojourn_time` ($1/(1 - P_{ii})$), `avg_time_to_others` (mean MFPT leaving state i), `avg_time_from_others` (mean MFPT arriving at state i).

mpt The underlying `net_mpt` object.

For a `netobject_group` the result is a "net_markov_stability_group": a named list holding one such object per group.

In `print.net_markov_stability()`: x, invisibly.

In `print.net_markov_stability_group()`: x invisibly.

In `plot.net_markov_stability()`: `plot.net_markov_stability` returns a faceted ggplot object when `combined = TRUE`, and (invisibly) a named list of single-metric ggplots, one per entry of metrics, when `combined = FALSE`.

In `summary.net_markov_stability()`: the per-state stability table (the `$stability` data frame: one row per state with `state`, `persistence`, `stationary_prob`, `return_time`, `sojourn_time`, `avg_time_to_others`, `avg_time_from_others`), after printing the attractor and the most persistent state.

References

Kemeny, J.G. and Snell, J.L. (1976). *Finite Markov Chains*. Springer-Verlag.

See Also

[passage_time](#)

Examples

```
net <- build_network(as.data.frame(trjectories), method = "relative")
ms <- markov_stability(net)
print(ms)
```

```
plot(ms)
```

mark_first_state	Mark leading-NA cells with an explicit state label
------------------	--

Description

Mirror of [mark_terminal_state\(\)](#) for *left-censored* sequence data. Replaces every cell *before* each row's first observed state with the label given by `state`. The resulting chain has a structurally *recurrent* "Start" state that everyone enters from - useful for cohort-entry analyses where students join at different time points and you want a uniform pre-observation marker.

Usage

```
mark_first_state(data, state = "Start", cols = NULL)
```

Arguments

<code>data</code>	A wide-format matrix or data.frame (rows = actors, cols = time steps) of state labels with NA for missing observations.
<code>state</code>	Character. Label to insert in leading-NA cells. Default "Start". If the label already occurs somewhere in data, the function warns and appends <code>_1</code> , <code>_2</code> , ... until it is unique, so the inserted marker is never confused with an observed state.
<code>cols</code>	Optional state-column names; otherwise all columns.

Details

Unlike [mark_terminal_state\(\)](#), the marked state is **not** absorbing in the resulting transition matrix - every transition from "Start" goes to one of the original states (the actor's first observed state), and the "Start" row is row-stochastic exactly as the data dictates.

Value

A character data.frame of the same shape as `data` (or of `data[cols]` when `cols` is given) with leading NAs filled by `state`. The label actually used is attached as the "leading_state" attribute, which matters when it had to be made unique.

See Also

[mark_terminal_state\(\)](#), [actor_endpoints\(\)](#)

Examples

```
M <- mark_first_state(trjectories, state = "Start")
# In a chain built from M, "Start" is a transient entry point.
```

mark_terminal_state	<i>Mark terminal-NA cells with an explicit state label</i>
---------------------	--

Description

Replaces every cell after each row's last observed state with the label given by `state`, leaving non-terminal NAs untouched. The result, passed to `build_network()`, yields a Markov chain in which the marked state is **absorbing** by construction ($P[\text{state}, \text{state}] = 1$).

Usage

```
mark_terminal_state(data, state = "End", cols = NULL)
```

Arguments

<code>data</code>	A wide-format matrix or <code>data.frame</code> (rows = actors, cols = time steps) of state labels with NA for missing observations.
<code>state</code>	Character. Label to insert in terminal-NA cells. Default "End". If the label already occurs somewhere in <code>data</code> , the function warns and appends <code>_1</code> , <code>_2</code> , ... until it is unique, so the absorbing marker is never confused with an observed state.
<code>cols</code>	Optional state-column names; otherwise all columns.

Details

This is the small piece of pre-processing required to turn right-censored sequence data into an absorbing-chain model. The chain on the resulting matrix has one extra state (`state`) which is structurally absorbing because every cell after the actor's last observed step has been set to `state` - the chain stays there forever once entered.

Use `chain_structure()` on the result to compute mean absorption time, absorption probabilities, and per-state classification. Note that `markov_stability()` is *not* the right summary for absorbing chains; its stationary distribution will collapse to the absorbing state.

Value

A character `data.frame` of the same shape as `data` (or of `data[cols]` when `cols` is given) with terminal NAs filled by `state`. The label actually used is attached as the `"terminal_state"` attribute, which matters when it had to be made unique.

See Also

`actor_endpoints()`, `chain_structure()`, `build_network()`

Examples

```
M <- mark_terminal_state(trjectories, state = "Dropout")
net <- build_network(M, method = "relative")
chain_structure(net)
```

mogen_transitions	<i>Extract Transition Table from a MOGen Model</i>
-------------------	--

Description

Returns a data frame of all transitions at a given Markov order, sorted by count (descending). Each row shows the full path as a readable sequence of states, along with the observed count and transition probability.

Usage

```
mogen_transitions(x, order = NULL, min_count = 1L)
```

Arguments

x	A <code>net_mogen</code> object from <code>build_mogen()</code> .
order	Integer ≥ 1 and at most the highest order tested. Which order's transitions to extract. Must be a whole number; a non-integer value is an error rather than being silently truncated. Defaults to the optimal order selected by the model - pass an explicit order when that optimal order is 0, which has no transition table and therefore errors.
min_count	Integer. Minimum observed count to include (default 1). Use this to filter out rare transitions that have unreliable probabilities.

Details

At order k , each edge in the De Bruijn graph represents a $(k+1)$ -step path. For example, at order 2, the edge from node "AI -> FAIL" to node "FAIL -> SOLVE" represents the three-step path AI -> FAIL -> SOLVE. The path column reconstructs this full sequence for readability.

Value

A data frame with one row per retained transition, sorted by count (descending), with columns:

path The full state sequence (e.g., "AI -> FAIL -> SOLVE").
count Number of times this transition was observed.
probability Transition probability $P(\text{to} | \text{from})$, rounded to 4 decimal places.
from The context / conditioning states (k -gram source node).
to The predicted next state.

A zero-row data frame with the same columns when no transition reaches `min_count`.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
mg <- build_mogen(seqs, max_order = 2)
mogen_transitions(mg, order = 1)

trajs <- list(c("A","B","C","D"), c("A","B","D","C"),
             c("B","C","D","A"), c("C","D","A","B"))
m <- build_mogen(trajs, max_order = 3)
mogen_transitions(m, order = 1)
```

mosaic_analysis

Two-variable mosaic analysis (chi-square test + flat mosaic)

Description

Analyses the association between two categorical columns of a `data.frame`. Builds the contingency table, drops sparse categories below `min_count`, runs a Pearson chi-square test (or Fisher's exact test), computes Cramer's V with a df-adjusted effect-size label, and draws a flat `ggplot2` mosaic whose tile area encodes counts and whose fill encodes the standardized Pearson residual (Nestimate diverging palette). All tabular output is a tidy one-row-per-cell `data.frame`.

Usage

```
mosaic_analysis(
  data,
  var1,
  var2,
  min_count = 10L,
  test = c("chisq", "fisher"),
  percentage_base = c("total", "row", "column"),
  tile_label = c("count", "percent", "residual", "category", "none"),
  title = "",
  ...
)

## S3 method for class 'mosaic_analysis'
plot(x, ...)

## S3 method for class 'mosaic_analysis'
print(x, ...)

## S3 method for class 'mosaic_analysis'
summary(object, ...)
```

Arguments

<code>data</code>	A data.frame containing the two variables.
<code>var1</code>	Character. Name of the first variable (mosaic columns).
<code>var2</code>	Character. Name of the second variable (stacked within columns).
<code>min_count</code>	Integer. Minimum marginal count for a category to be kept. Categories of either variable below this are dropped before testing. Default 10.
<code>test</code>	Character. "chisq" (default) Pearson chi-square, or "fisher" Fisher's exact test (simulated p-value). Cramer's V and the residual fill are always derived from the chi-square statistic.
<code>percentage_base</code>	Character. Base for the "percent" tile label and the pct column: "total" (default), "row" (within var1), or "column" (within var2).
<code>tile_label</code>	Character. What to print inside each tile: "count" (default), "percent", "residual", "category" (var2 level), or "none".
<code>title</code>	Character. Plot title. Default "".
<code>...</code>	Further flat-mosaic styling arguments passed to the renderer (e.g. <code>col_label_side</code> , <code>row_label_side</code> , <code>legend_position</code> , <code>legend_size</code> , <code>label_size</code> , <code>palette</code>). Tile fill uses the ColorBrewer RdBu ramp by default (override with <code>palette</code>). Column labels auto-rotate to vertical when there are more than 6 columns; pass <code>col_label_angle</code> to force an angle. In <code>plot.mosaic_analysis()</code> : Styling overrides forwarded to the flat renderer. In <code>print.mosaic_analysis()</code> and <code>summary.mosaic_analysis()</code> : Ignored.
<code>x</code>	For the <code>plot()</code> and <code>print()</code> methods: an object of class <code>mosaic_analysis</code> .
<code>object</code>	For the <code>summary()</code> method: an object of class <code>mosaic_analysis</code> .

Value

An object of class "mosaic_analysis": a list with

plot The flat mosaic ggplot object.

counts Tidy data.frame, one row per (var1, var2) cell, with observed, expected, residual (standardized), and pct (on percentage_base).

stats One-row data.frame: test, statistic, df, p_value, cramers_v, effect_size, n.

test The raw htest object.

cramers_v, effect_size Effect size value and label.

table The filtered contingency table.

removed List of dropped var1 / var2 categories.

n_original, n_filtered Row counts before/after filtering.

vars Named character vector `c(var1 = , var2 = )`.

plot_parts, plot_args The residual matrix, table, and styling arguments retained so `plot()` can re-render without re-testing.

Use `print()` for the test summary and `summary()` for the tidy per-cell table.

In `plot.mosaic_analysis()`: The re-rendered flat mosaic ggplot object, invisibly; the plot is drawn on the active device as a side effect.

In `print.mosaic_analysis()`: x, invisibly.

In `summary.mosaic_analysis()`: The tidy per-cell data.frame: one row per (var1, var2) cell, with the two variable columns (named after var1 / var2) plus observed, expected, residual and pct. The one-row test summary is attached as the "stats" attribute.

Methods

- `plot.mosaic_analysis()`: Re-renders the flat mosaic from the stored contingency table and residuals, so styling can be changed without re-running the test. Any flat-mosaic styling argument (`tile_label`, `pct_base`, `col_label_side`, `legend_size`, ...) may be overridden via

See Also

[mosaic_plot](#) for the network/table mosaic (which also accepts `style = "flat"`).

Examples

```
data(group_regulation_long, package = "Nestimate")
res <- mosaic_analysis(group_regulation_long, "Course", "Action",
                      min_count = 20)

res
head(summary(res))

plot(res, tile_label = "percent")
```

mosaic_plot

Mosaic Plot of a Network's Transition or Co-occurrence Counts

Description

Draws a Hartigan-Friendly mosaic (marimekko geometry, chi-square standardized-residual fill) for an integer-weighted network. Equivalent in algorithm and appearance to `tna::plot_mosaic()`; named differently to avoid an export clash when both packages are attached.

Usage

```
mosaic_plot(x, ...)

## Default S3 method:
mosaic_plot(x, ...)

## S3 method for class 'netobject'
```

```
mosaic_plot(  
  x,  
  xlab = NULL,  
  ylab = NULL,  
  range = NULL,  
  top_angle = NULL,  
  left_angle = NULL,  
  residuals = c("permutation", "asymptotic"),  
  n_perm = 500L,  
  seed = NULL,  
  values = FALSE,  
  style = c("classic", "flat"),  
  ...  
)
```

```
## S3 method for class 'htna'  
mosaic_plot(  
  x,  
  xlab = NULL,  
  ylab = NULL,  
  range = NULL,  
  top_angle = NULL,  
  left_angle = NULL,  
  residuals = c("permutation", "asymptotic"),  
  n_perm = 500L,  
  seed = NULL,  
  values = FALSE,  
  style = c("classic", "flat"),  
  ...  
)
```

```
## S3 method for class 'mcm1'  
mosaic_plot(  
  x,  
  level = c("macro", "clusters"),  
  xlab = NULL,  
  ylab = NULL,  
  range = NULL,  
  top_angle = NULL,  
  left_angle = NULL,  
  residuals = c("permutation", "asymptotic"),  
  n_perm = 500L,  
  seed = NULL,  
  ncol = 2L,  
  values = FALSE,  
  style = c("classic", "flat"),  
  ...  
)
```

```
## S3 method for class 'netobject_group'
mosaic_plot(
  x,
  xlab = NULL,
  ylab = NULL,
  range = NULL,
  top_angle = NULL,
  left_angle = NULL,
  residuals = c("permutation", "asymptotic"),
  n_perm = 500L,
  seed = NULL,
  ncol = 2L,
  values = FALSE,
  style = c("classic", "flat"),
  ...
)

## S3 method for class 'table'
mosaic_plot(
  x,
  xlab = "Row",
  ylab = "Column",
  range = NULL,
  top_angle = NULL,
  left_angle = NULL,
  residuals = c("permutation", "asymptotic"),
  n_perm = 500L,
  seed = NULL,
  values = FALSE,
  style = c("classic", "flat"),
  ...
)

## S3 method for class 'matrix'
mosaic_plot(x, ...)
```

Arguments

<code>x</code>	One of the four data-bearing Nestimate classes: <code>netobject</code> (single mosaic of <code>\$weights</code>), <code>netobject_group</code> (one panel per group), <code>mcml</code> (between-cluster mosaic by default; per-cluster panels with <code>level = "within"</code>), or <code>htna</code> (single mosaic of <code>\$weights</code> ; <code>htna</code> inherits <code>netobject</code> so the geometry matches). Also accepts a contingency table or plain numeric matrix for ad-hoc plotting.
<code>...</code>	Flat styling overrides forwarded to the flat renderer when <code>style = "flat"</code> (otherwise ignored).
<code>xlab, ylab</code>	Axis labels. <code>NULL</code> (default) draws no axis title on the four data-bearing methods; the <code>table</code> / <code>matrix</code> methods default to <code>"Row"</code> and <code>"Column"</code> . Pass any string to

	set one.
range	Numeric of length 2 giving the lower and upper colour-scale limits for the standardized residual. NULL (default) auto-fits the limits to the symmetric range $c(-M, M)$ where $M = \max(stdres)$ floored at 1, so no signal is squished and the legend stays readable on a near-independent table. Pass an explicit range (e.g. $c(-4, 4)$ for tna-style display, $c(-6, 6)$ for moderate clipping) to clamp the colour scale.
top_angle, left_angle	Rotation in degrees for the top (x) and left (y) tick labels. NULL (default) uses the auto rule 90 if <code>n_levels > 3</code> else 0 on each axis. Pass any numeric to override (e.g. <code>top_angle = 45</code> , <code>left_angle = 0</code>).
residuals	One of "permutation" (default) or "asymptotic". "permutation" computes empirical-null z-scores by shuffling one variable's labels against the other for <code>n_perm</code> draws and reporting $(0 - \text{mean_perm}) / \text{sd_perm}$ per cell. Robust on sparse tables. "asymptotic" returns <code>stats::chisq.test()\$stdres</code> (the closed-form $(0 - E) / \sqrt{E * (1 - p_row) * (1 - p_col)}$) that <code>vcd</code> and <code>tna</code> use).
n_perm	Number of permutations when <code>residuals = "permutation"</code> . Default 500; use ≥ 1000 for stable tail estimates.
seed	Optional integer seed for the permutation RNG. Use for reproducible plots; ignored when <code>residuals = "asymptotic"</code> .
values	Logical. When TRUE, overlay each cell's standardized residual as a numeric label (one decimal). Text colour switches to white on saturated cells (<code>lstdres > 1.5</code>) and dark grey otherwise. Default FALSE – the colour bar legend already conveys the sign and magnitude.
style	Character. "classic" (default) draws the black-bordered theme_minimal mosaic with axis labels; "flat" draws the flat theme_void mosaic with white gutters, a soft background and a colour-bar legend. With <code>style = "flat"</code> , <code>values = TRUE</code> prints the standardized residual inside each tile, and extra flat styling arguments (<code>tile_label</code> , <code>col_label_side</code> , <code>row_label_side</code> , <code>legend_position</code> , <code>legend_size</code> , <code>label_size</code> , ...; see mosaic_analysis) may be passed via Not supported for multi-panel <code>mcml</code> (<code>level = "clusters"</code>), which falls back to the classic faceted style.
level	For <code>mcml</code> only. "macro" (default) draws a single mosaic of <code>x\$macro\$weights</code> (the cluster-by-cluster aggregate); "clusters" draws one mosaic per cluster from <code>x\$clusters[[k]]\$weights</code> , faceted into one combined ggplot.
ncol	Number of columns in the multi-panel layout. Default 2. Effective only for <code>mcml</code> with <code>level = "clusters"</code> , the one multi-panel case; <code>netobject_group</code> is drawn as a single (group x state) mosaic, so <code>ncol</code> is accepted but has no effect there.

Details

Column widths are proportional to row marginals of the weight matrix (incoming totals when the matrix is transposed, as for transitions). Within each column, segment heights are proportional to that row's conditional distribution. Cell fill is the standardized residual under the independence null – a permutation z-score by default, or the closed-form `stats::chisq.test()` residual with

residuals = "asymptotic" (see residuals) – on a diverging palette whose limits auto-fit the observed residuals unless range is supplied. Mosaics need integer counts: when \$weights is already integer (method = "frequency" / "co_occurrence") it is used directly; for a single netobject / htma otherwise (relative, glasso, cor, ...) order-1 transition counts are recounted from the raw \$data sequences. The function errors only when neither integer weights nor \$data are available.

Value

A ggplot object: one geom_rect layer with one rectangle per contingency-table cell, filled by the standardized residual. mcm1 with level = "clusters" returns a single facet_wrap-ed ggplot (one panel per cluster, shared fill scale); every other input returns a single-panel ggplot.

See Also

[plot_mosaic](#) for the lower-level data.frame primitive.

Examples

```
data(group_regulation_long, package = "Nestimate")
net <- build_network(group_regulation_long, method = "frequency",
                    format = "long", actor = "Actor", action = "Action",
                    order = "Time")
mosaic_plot(net, seed = 1)
```

nct

Network Comparison Test

Description

Tests whether two networks estimated from independent samples differ at three levels: **global strength** (M-statistic), **network structure** (S-statistic, max absolute edge difference), and **individual edges** (E-statistic per edge). Inference is via permutation of group labels.

Usage

```
nct(
  data1,
  data2,
  iter = 1000L,
  gamma = 0.5,
  paired = FALSE,
  abs = TRUE,
  weighted = TRUE,
  p_adjust = "none"
)

## S3 method for class 'net_nct'
print(x, ...)
```

```
## S3 method for class 'net_nct'
summary(object, ...)
```

Arguments

<code>data1</code>	A numeric matrix or data.frame of observations from group 1.
<code>data2</code>	A numeric matrix or data.frame of observations from group 2. Same number of columns as <code>data1</code> .
<code>iter</code>	Integer. Number of permutation iterations. Default 1000.
<code>gamma</code>	EBIC tuning parameter for glasso. Default 0.5.
<code>paired</code>	Logical. If TRUE, perform a paired permutation (within-subject swap). Default FALSE.
<code>abs</code>	Logical. If TRUE, compute global strength on absolute edge weights. Default TRUE.
<code>weighted</code>	Logical. If TRUE, use weighted networks for the tests. If FALSE, binarize before computing statistics. Default TRUE.
<code>p_adjust</code>	P-value adjustment method for the per-edge tests (any method in <code>stats::p.adjust.methods</code>). Default "none".
<code>x</code>	For the <code>print()</code> method: an object of class <code>net_nct</code> .
<code>...</code>	In <code>print.net_nct()</code> and <code>summary.net_nct()</code> : Ignored.
<code>object</code>	For the <code>summary()</code> method: an object of class <code>net_nct</code> .

Details

Follows `NetworkComparisonTest::NCT()` with defaults `abs = TRUE`, `weighted = TRUE`, `paired = FALSE`. The network estimator is EBIC-selected glasso applied to a Pearson correlation matrix, with `Matrix::nearPD` symmetrization (matching NCT's `NCT_estimator_GGM` default). The glasso solver is not the Fortran one NCT wraps, so results agree to independent-solver precision (of the order of $1e-4$ on the test statistics) rather than bit-for-bit, even under the same seed.

Value

A list of class `net_nct` with elements:

nw1, nw2 Estimated weighted adjacency matrices.

M List with observed, perm, `p_value` for the global strength test. P-values are permutation p-values, $(\text{sum}(\text{perm} \geq \text{observed}) + 1) / (\text{iter} + 1)$.

S Same structure for the maximum absolute edge difference.

E Same structure for the per-edge tests (observed and `p_value` are one value per upper-triangle edge, perm an `iter` by edges matrix), plus `edge_names`, a two-column data frame of the node pairs (NULL when `data1` has no column names).

n_iter Number of permutations.

paired Whether a paired test was used.

params List of the settings used: `gamma`, `abs`, `weighted`, `p_adjust`.

In `print.net_nct()`: The input object, invisibly.

In `summary.net_nct()`: A data frame with columns `from`, `to`, `diff_observed`, `p_value`, `significant`. Attributes `m_stat` and `s_stat` each hold a one-row data frame with observed and `p_value`.

Methods

- `summary.net_nct()`: Returns a tidy data frame with one row per edge test. The global M (strength) and S (structure) statistics are attached as attributes.

Examples

```
set.seed(1)
x1 <- matrix(rnorm(100 * 4), 100, 4)
x2 <- matrix(rnorm(100 * 4), 100, 4)
colnames(x1) <- colnames(x2) <- paste0("V", 1:4)
# iter = 20 keeps the example fast; a real analysis uses 1000 or more.
res <- nct(x1, x2, iter = 20)
res
summary(res)
```

network_reliability *Split-Half Reliability for Network Estimates*

Description

Assesses the stability of network estimates by repeatedly splitting sequences into two halves, building networks from each half, and comparing them. Supports single-model reliability assessment and multi-model comparison with optional scaling for cross-method comparability.

For transition methods ("relative", "frequency", "co_occurrence"), uses pre-computed per-sequence count matrices for fast resampling (same infrastructure as [bootstrap_network](#)).

Usage

```
network_reliability(
  ...,
  iter = 1000L,
  split = 0.5,
  scale = "none",
  seed = NULL
)

## S3 method for class 'net_reliability'
print(x, ...)

## S3 method for class 'net_reliability'
summary(object, ...)

## S3 method for class 'net_reliability'
plot(x, bins = 60L, combined = TRUE, ...)
```

Arguments

<code>...</code>	One or more netobjects (from build_network). If unnamed, each model is auto-named from its <code>\$method</code> ; duplicate names are made unique with <code>make.unique()</code> . A <code>netobject_group</code> is flattened into its constituent models (named by group), and an <code>mcml</code> or <code>cograph_network</code> is converted first. In <code>plot.net_reliability()</code> and <code>print.net_reliability()</code> : Additional arguments (ignored). In <code>summary.net_reliability()</code> : Ignored.
<code>iter</code>	Integer. Number of split-half iterations (default: 1000).
<code>split</code>	Numeric. Fraction of sequences assigned to the first half (default: 0.5).
<code>scale</code>	Character. Scaling applied to both split-half matrices before computing metrics. One of "none" (default), "minmax", "standardize", or "proportion". Use scaling when comparing models on different scales (e.g. frequency vs relative).
<code>seed</code>	Integer or NULL. RNG seed for reproducibility.
<code>x</code>	For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_reliability</code> .
<code>object</code>	For the <code>summary()</code> method: an object of class <code>net_reliability</code> .
<code>bins</code>	Integer. Number of histogram bins per panel (default 60).
<code>combined</code>	When TRUE (default), all four metrics are shown in one ggplot via <code>facet_wrap(~metric)</code> . When FALSE, returns a named list of four single-panel ggplots, one per metric.

Value

An object of class "net_reliability" containing:

iterations Data frame with columns `model`, `mean_dev`, `median_dev`, `cor`, `max_dev` (one row per iteration per model).

summary Data frame with columns `model`, `metric`, `mean`, `sd`.

models Named list of the original netobjects.

iter Number of iterations.

split Split fraction.

scale Scaling method used.

In `print.net_reliability()`: The input object, invisibly.

In `summary.net_reliability()`: A tidy data frame with columns `model`, `metric`, `mean`, `sd` summarising the split-half iterations.

In `plot.net_reliability()`: A ggplot object (invisibly), or a named list of four ggplots when `combined = FALSE`.

Methods

- `plot.net_reliability()`: Density plots of split-half metrics faceted by metric type. Multi-model comparisons show overlaid densities colored by model.

See Also

[build_network](#), [bootstrap_network](#)

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),
  V2 = c("B","C","A","B")), method = "relative")
rel <- network_reliability(net, iter = 10)

set.seed(1)
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
rel <- network_reliability(net, iter = 100, seed = 42)
print(rel)
```

net_aggregate_weights *Aggregate Edge Weights*

Description

Aggregates a vector of edge weights using various methods. Compatible with igraph's edge.attr.comb parameter.

Usage

```
net_aggregate_weights(w, method = "sum", n_possible = NULL)
```

Arguments

w	Numeric vector of finite edge weights. NA and zero weights are excluded before aggregation, so every method (including "density", "min", "max", "prod", "geomean") operates on the non-zero, non-NA subset.
method	Single aggregation method: "sum", "mean", "median", "max", "min", "prod", "density", or "geomean". Because zeros are stripped first, "density" ($\text{sum}(w) / n_possible$) and "mean" ($\text{sum}(w) / \text{number of non-zero edges}$) return the <i>same</i> value whenever the block is fully dense – i.e. when the count of non-zero edges equals n_possible. They diverge only when zero/NA edges are present (then "density" divides by the larger n_possible, "mean" by the smaller non-zero count).
n_possible	Optional single finite numeric number of possible edges for density calculation. When omitted, "density" falls back to $\text{sum}(w) / \text{length}(w)$ on the non-zero subset (equivalent to "mean"); supply n_possible (e.g. the block size $n_i * n_j$) for a true edge density.

Value

A single numeric value: the chosen aggregation of the non-zero, non-NA weights, or 0 when none remain.

Examples

```
w <- c(0.5, 0.8, 0.3, 0.9)
net_aggregate_weights(w, "sum") # 2.5
net_aggregate_weights(w, "mean") # 0.625
net_aggregate_weights(w, "max") # 0.9
net_aggregate_weights(w, "density", n_possible = 9) # 2.5 / 9
```

net_centrality

Compute Centrality Measures for a Network

Description

Computes centrality measures from a `netobject`, `netobject_group`, `mcml`, or `cograph_network`. The built-in measures match `tna::centralities()` without importing `tna` or `igraph`: strength is taken from the weight matrix directly, and the path-based measures (betweenness, closeness) come from all-pairs shortest paths computed in-package by Floyd-Warshall. The only intentional default difference from `tna` is that Diffusion is range-normalized by default.

Usage

```
net_centrality(
  x,
  measures = NULL,
  loops = FALSE,
  normalize = FALSE,
  invert = TRUE,
  normalize_diffusion = TRUE,
  centrality_fn = NULL,
  ...
)

## S3 method for class 'net_centrality'
plot(
  x,
  reorder = TRUE,
  ncol = 3L,
  type = c("bar", "line", "heatmap"),
  scales = c("free_x", "fixed"),
  profile_scale = c("measure", "none"),
  labels = TRUE,
  drop_zero = FALSE,
  ...
)

## S3 method for class 'net_centrality_group'
plot(
  x,
```

```

reorder = TRUE,
ncol = 3L,
type = c("bar", "line", "delta"),
scales = c("free_x", "fixed"),
palette = "Set2",
profile_scale = c("measure", "none"),
labels = FALSE,
drop_zero = FALSE,
...
)

```

Arguments

<code>x</code>	A <code>netobject</code> , <code>netobject_group</code> , <code>mcml</code> , or <code>cograph_network</code> . For the <code>plot()</code> method: an object of class <code>net_centrality</code> or <code>net_centrality_group</code> .
<code>measures</code>	Character vector. Centrality measures to compute. Defaults to <code>c("InStrength", "Betweenness", "Diffusion")</code> . Pass <code>"all"</code> for every built-in measure: <code>"OutStrength", "InStrength", "ClosenessIn", "ClosenessOut", "Closeness", "Betweenness", "BetweennessRSP", "Diffusion",</code> and <code>"Clustering"</code> . The legacy aliases <code>"InCloseness"</code> and <code>"OutCloseness"</code> are also accepted.
<code>loops</code>	Logical. Include self-loops (diagonal) in computation? Default: <code>FALSE</code> .
<code>normalize</code>	Logical. Range-normalize all requested measures using the same transformation as <code>tna::centralities(normalize = TRUE)</code> . Default: <code>FALSE</code> .
<code>invert</code>	Logical. Invert weights for shortest-path measures? Default: <code>TRUE</code> , matching <code>tna</code> .
<code>normalize_diffusion</code>	Logical. Range-normalize Diffusion even when <code>normalize = FALSE</code> . Default: <code>TRUE</code> .
<code>centrality_fn</code>	Optional function. Custom centrality function that takes a weight matrix and returns a named list of centrality vectors.
<code>...</code>	Additional arguments (ignored). In <code>plot.net_centrality()</code> and <code>plot.net_centrality_group()</code> : Additional arguments ignored.
<code>reorder</code>	In <code>plot.net_centrality()</code> : Logical. Reorder states within each centrality panel by centrality value. Default: <code>TRUE</code> . In <code>plot.net_centrality_group()</code> : Logical. Reorder states by their mean value within each centrality panel. Default: <code>TRUE</code> .
<code>ncol</code>	Integer. Number of facet columns. Default: 3.
<code>type</code>	In <code>plot.net_centrality()</code> : Plot type. <code>"bar"</code> shows one faceted horizontal bar chart per measure; <code>"line"</code> shows state profiles as lines across measures (the value <code>"profile"</code> is still accepted as an alias); <code>"heatmap"</code> shows a states-by-measures tile grid, each measure scaled to 0–1 for cross-measure comparability with the raw value printed in the tile. Default: <code>"bar"</code> . In <code>plot.net_centrality_group()</code> : Plot type. <code>"bar"</code> shows grouped bars within each measure; <code>"line"</code> facets by state and draws one line per group across centrality measures (<code>"profile"</code> is accepted as an alias); <code>"delta"</code> draws a diverging bar of group differences. With two groups it is the per-state difference (second group minus first); with three or

	more groups it is each group's deviation from the per-state group mean, so the largest gaps stand out either way. Default: "bar".
scales	Facet scale mode. "free_x" (default) uses free centrality axes; "fixed" keeps a common centrality axis.
profile_scale	Scaling used by type = "line". "measure" (default) rescales each centrality measure to 0–1 before drawing cross-measure profiles; "none" uses raw values.
labels	In plot.net_centrality(): Logical. Add compact value labels. Default: TRUE. In plot.net_centrality_group(): Logical. Add compact value labels. Default: FALSE.
drop_zero	In plot.net_centrality(): Logical. Drop measures whose values are all (near) zero so empty panels do not waste space. Default: FALSE (every requested measure is shown). In plot.net_centrality_group(): Logical. Drop measures whose values are all (near) zero so empty panels do not waste space. Default: FALSE.
palette	Brewer palette for groups. Default: "Set2".

Value

For a netobject or cograph_network: a net_centrality data frame, one row per node, with a state column and one further column per requested measure (node names are also the row names). For a netobject_group or an mcml: a net_centrality_group list of such data frames, one per group.

In plot.net_centrality() and plot.net_centrality_group(): A ggplot object.

References

- Freeman, L. C. (1978). Centrality in social networks: conceptual clarification. *Social Networks*, 1(3), 215–239. (betweenness, closeness)
- Opsahl, T., Agneessens, F. & Skvoretz, J. (2010). Node centrality in weighted networks: generalizing degree and shortest paths. *Social Networks*, 32(3), 245–251. (weighted strength and geodesics)
- Kivimäki, I., Lebichot, B., Saramäki, J. & Saerens, M. (2016). Two betweenness centrality measures based on randomized shortest paths. *Scientific Reports*, 6, 19668. (BetweennessRSP)
- Banerjee, A., Chandrasekhar, A. G., Duflo, E. & Jackson, M. O. (2013). The diffusion of microfinance. *Science*, 341(6144), 1236498. (Diffusion)
- Onnela, J.-P., Saramäki, J., Kertész, J. & Kaski, K. (2005). Intensity and coherence of motifs in weighted complex networks. *Physical Review E*, 71, 065103. (Clustering)

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),
  V3 = c("C", "A", "C", "B"))
net <- build_network(seqs, method = "relative")
net_centrality(net)
```

net_deprune

Undo Network Pruning

Description

Restores the original (pre-pruning) weights of a network pruned by [net_prune](#), without recomputation. The pruning record is kept, so [net_reprune](#) can re-apply it.

Usage

```
net_deprune(x, ...)
```

```
## S3 method for class 'netobject'
```

```
net_deprune(x, ...)
```

```
## S3 method for class 'netobject_group'
```

```
net_deprune(x, ...)
```

```
## Default S3 method:
```

```
net_deprune(x, ...)
```

Arguments

x	A pruned netobject or netobject_group.
...	Ignored.

Value

The network (or group) with original weights restored and its pruning marked inactive.

See Also

[net_prune](#), [net_reprune](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"),
                   V3 = c("C", "A", "B"))
net    <- build_network(seqs, method = "relative")
pruned <- net_prune(net, threshold = 0.2)
net_deprune(pruned)
```

 net_edge_betweenness *Edge Betweenness Network*

Description

Builds a network in which each edge's weight is replaced by its betweenness: the number of shortest paths between all node pairs that traverse that edge (fractional when shortest paths tie). This is the Nestimate counterpart of `tna::betweenness_network()` and produces identical values for transition networks; the name differs to avoid a clash with `tna::betweenness_network()` and `igraph::edge_betweenness()`.

Usage

```
net_edge_betweenness(x, invert = TRUE, ...)

## S3 method for class 'netobject'
net_edge_betweenness(x, invert = TRUE, ...)

## S3 method for class 'netobject_group'
net_edge_betweenness(x, invert = TRUE, ...)

## Default S3 method:
net_edge_betweenness(x, invert = TRUE, ...)

## S3 method for class 'net_edge_betweenness'
plot(x, style = c("bar", "forest", "delta"), top_n = NULL, labels = TRUE, ...)
```

Arguments

<code>x</code>	A <code>netobject</code> or <code>netobject_group</code> . For the <code>plot()</code> method: an object of class <code>net_edge_betweenness</code> .
<code>invert</code>	Logical. Invert weights to distances by $1/w$ before computing shortest paths? Default <code>TRUE</code> (correct for probability and frequency networks).
<code>...</code>	Additional arguments (ignored). In <code>plot.net_edge_betweenness()</code> : Additional arguments (ignored).
<code>style</code>	Plot style. "bar" (default) draws one horizontal bar per edge; "forest" draws a forest/lollipop chart (a stem from zero to a point) with a dashed reference line at the mean betweenness; "delta" draws each edge's deviation from the mean edge betweenness as a diverging bar (above the mean in blue, below in red).
<code>top_n</code>	Integer or <code>NULL</code> . Keep only the <code>top_n</code> highest edges. Default <code>NULL</code> (all edges with non-zero betweenness).
<code>labels</code>	Logical. Print the betweenness value beside each edge. Default <code>TRUE</code> .

Details

For a probability/transition network the edge weights are transition probabilities, so they are inverted to distances (`invert = TRUE`) before path-finding: the geodesic between two states is then the most probable route rather than the one with the fewest hops. Pass `invert = FALSE` when the weights already represent distances.

Directedness is taken from the network itself. A directed network yields an asymmetric betweenness matrix; an undirected (symmetric) network yields a symmetric one.

Value

For a `netobject`: a new network of class `c("net_edge_betweenness", "netobject", "cograph_network")` whose `$weights` are the edge-betweenness scores, with `method = "edge_betweenness"`. Call `extract_edges()` on it for a tidy per-edge table, or `plot()` to render it. The object preserves source-network metadata so [permutation](#) can test edge-betweenness differences by permuting the source networks. For a `netobject_group`: a `netobject_group` of such networks, one per group.

In `plot.net_edge_betweenness()`: A `ggplot` object.

Methods

- `plot.net_edge_betweenness()`: Draws the edges of a [net_edge_betweenness](#) network ranked by their betweenness, as a horizontal bar chart. This is the tidy, cograph-free companion to the node-link diagram: render the diagram with `cograph::splot(eb)` and the ranking with `plot(eb)`.

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),
  V3 = c("C", "A", "C", "B"))
net <- build_network(seqs, method = "relative")
eb <- net_edge_betweenness(net)
extract_edges(eb)
```

net_prune

Prune a Network's Edges

Description

Removes weak or non-significant edges from a network, keeping a record so the operation can be reversed. This is `Nestimate`'s counterpart of `tna::prune()`; the `net_` prefix avoids a name clash with `tna::prune()`.

Usage

```
net_prune(  
  x,  
  method = "threshold",  
  threshold = 0.1,  
  lowest = 0.05,  
  level = 0.5,  
  boot = NULL,  
  ...  
)  
  
## S3 method for class 'netobject'  
net_prune(  
  x,  
  method = "threshold",  
  threshold = 0.1,  
  lowest = 0.05,  
  level = 0.5,  
  boot = NULL,  
  ...  
)  
  
## S3 method for class 'netobject_group'  
net_prune(  
  x,  
  method = "threshold",  
  threshold = 0.1,  
  lowest = 0.05,  
  level = 0.5,  
  boot = NULL,  
  ...  
)  
  
## Default S3 method:  
net_prune(  
  x,  
  method = "threshold",  
  threshold = 0.1,  
  lowest = 0.05,  
  level = 0.5,  
  boot = NULL,  
  ...  
)
```

Arguments

x	A netobject or netobject_group.
method	One of "threshold", "lowest", "disparity", "bootstrap". Default "threshold".

threshold	Numeric cut-off for method = "threshold". Default 0.1.
lowest	Quantile (0-1) for method = "lowest". Default 0.05.
level	Significance level (0-1) for method = "disparity". Default 0.5.
boot	Optional precomputed net_bootstrap for method = "bootstrap".
...	Passed to bootstrap_network when method = "bootstrap".

Details

Pruning is non-destructive: the pruned network carries a "pruning" attribute holding the original weights, the pruned weights, the parameters used, and a tidy table of removed edges. Use [net_deprune](#) to restore the original weights and [net_reprune](#) to re-apply the pruning, both without recomputation. [net_pruning_details](#) reports what was removed.

Methods:

"threshold" Remove edges with weight $\leq$ threshold.

"lowest" Remove the lowest lowest quantile of non-zero edges.

"disparity" Serrano disparity-filter backbone at significance level.

"bootstrap" Remove edges deemed non-significant by [bootstrap_network](#) (pass a precomputed result via boot, or extra bootstrap arguments via ...).

For "threshold", "lowest", and "disparity" an edge is dropped only when its removal leaves the network weakly connected.

Diagonal self-loops (self-transitions) are observed data: they are counted equally when computing the cut-off but are never removed by any method. (This is a deliberate divergence from `tna::prune()`, which prunes self-loops like any other edge.)

Value

The input network (or group) with pruned \$weights and a "pruning" attribute. Class is unchanged.

See Also

[net_deprune](#), [net_reprune](#), [net_pruning_details](#)

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "A", "C", "B"), V2 = c("B", "C", "B", "A", "C"),
  V3 = c("C", "A", "C", "B", "A"))
net <- build_network(seqs, method = "relative")
pruned <- net_prune(net, method = "threshold", threshold = 0.2)
net_pruning_details(pruned)
```

net_pruning_details	<i>Report Network Pruning Details</i>
---------------------	---------------------------------------

Description

Returns the edges removed by [net_prune](#) as a tidy one-row-per-edge data frame, with the method, cut-off, and retained/removed counts attached as attributes and shown by its print method.

Usage

```
net_pruning_details(x, ...)

## S3 method for class 'netobject'
net_pruning_details(x, ...)

## S3 method for class 'netobject_group'
net_pruning_details(x, ...)

## Default S3 method:
net_pruning_details(x, ...)

## S3 method for class 'net_pruning_details'
print(x, ...)
```

Arguments

x	A pruned netobject or netobject_group. For the print() method: an object of class net_pruning_details.
...	Ignored. In print.net_pruning_details(): Ignored.

Value

For a netobject: a net_pruning_details data frame (columns from, to, weight) of removed edges. For a netobject_group: a named list of such data frames.

In print.net_pruning_details(): x, invisibly.

See Also

[net_prune](#)

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "A", "C", "B"), V2 = c("B", "C", "B", "A", "C"),
  V3 = c("C", "A", "C", "B", "A"))
net <- build_network(seqs, method = "relative")
net_pruning_details(net_prune(net, threshold = 0.2))
```

`net_reprune`*Re-apply Network Pruning*

Description

Re-applies a previously computed pruning that was undone by [net_deprune](#), without recomputation.

Usage

```
net_reprune(x, ...)  
  
## S3 method for class 'netobject'  
net_reprune(x, ...)  
  
## S3 method for class 'netobject_group'  
net_reprune(x, ...)  
  
## Default S3 method:  
net_reprune(x, ...)
```

Arguments

<code>x</code>	A depruned <code>netobject</code> or <code>netobject_group</code> .
<code>...</code>	Ignored.

Value

The network (or group) with pruned weights re-applied and its pruning marked active.

See Also

[net_prune](#), [net_deprune](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"),  
                  V3 = c("C", "A", "B"))  
net    <- build_network(seqs, method = "relative")  
pruned <- net_prune(net, threshold = 0.2)  
undone <- net_deprune(pruned)  
net_reprune(undone)
```

outcome_model

*Model Unit-Level Outcomes from Sequence or Network Predictors***Description**

Fits a regression of an outcome on predictor columns – pattern indicators, topological features from [simplicial_features](#), or any other numeric covariates – and returns a tidy effect table with confidence intervals and multiplicity-corrected p-values.

Usage

```
outcome_model(
  data,
  outcome,
  predictors,
  group = NULL,
  adjust = NULL,
  family = c("auto", "binomial", "gaussian"),
  select = c("none", "split"),
  n_select = 10L,
  correction = "BH",
  ci_level = 0.95,
  seed = NULL
)

## S3 method for class 'net_outcome_model'
print(x, ...)

## S3 method for class 'net_outcome_model'
summary(object, ...)

## S3 method for class 'net_outcome_model'
plot(x, ...)
```

Arguments

data	A <code>data.frame</code> with one row per unit of analysis.
outcome	Name of the outcome column. A two-valued outcome is modelled with a binomial family, a numeric one with gaussian; override with <code>family</code> .
predictors	Character vector of predictor column names.
group	Optional column name giving a grouping factor. When supplied and lme4 is installed, a random intercept per group is added – the right treatment for units nested in actors. lme4 is a suggested package: when it is not installed the random intercept is dropped, a <code>"nestimate_no_lme4"</code> warning is raised, and a plain glm is fitted instead.

adjust	Optional character vector of covariates entered before the predictors. Use it for exposure: a unit observed longer contains more of every pattern, so an unadjusted effect can be volume in disguise.
family	"auto" (default), "binomial" or "gaussian".
select	"none" (default) fits all predictors; "split" ranks them on half the data and fits on the other half.
n_select	Number of predictors kept when select = "split". Default 10.
correction	Multiplicity correction passed to p.adjust . Default "BH".
ci_level	Confidence level for the intervals. Default 0.95.
seed	Optional integer seed for the split, so the result is reproducible. The RNG state is restored on exit.
x	For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_outcome_model</code> .
...	In <code>plot.net_outcome_model()</code> and <code>summary.net_outcome_model()</code> : Ignored. In <code>print.net_outcome_model()</code> : Unused.
object	For the <code>summary()</code> method: an object of class <code>net_outcome_model</code> .

Value

An object of class `net_outcome_model`: a list whose `$effects` element is the tidy data frame, one row per model term (the intercept included), with columns `term`, `estimate`, `std_error`, `statistic`, `ci_lower`, `ci_upper`, `p_value` and `p_adj` (NA on the intercept row, which is excluded from the correction), plus `odds_ratio`, `or_lower` and `or_upper` for a binomial fit. The remaining elements are the fitted `$model` (a `glm`, or an **lme4** fit when a random intercept was added), `$family`, `$n` (rows the reported model was fitted on), `$n_groups` (NA unless mixed), `$selected`, `$dropped` (zero-variance predictors), `$adjust`, `$select`, `$correction`, `$ci_level`, `$mixed` and `$outcome`. Retrieve the table with [effects_table](#).

In `print.net_outcome_model()`: `print` returns its input invisibly.

In `summary.net_outcome_model()`: `summary` returns the tidy effect table.

In `plot.net_outcome_model()`: `plot` returns a `ggplot` forest of the effects.

Honest inference

Choosing predictors by their association with the outcome and then testing them on the same rows invalidates the p-values. With `select = "split"` the data is halved: predictors are ranked on one half and the reported model is fitted on the other, so the returned inference is valid for the selected set. `select = "none"` (default) fits every supplied predictor and needs no split.

See Also

[simplicial_features](#), [effects_table](#)

Examples

```

set.seed(1)
d <- data.frame(
  hint    = rbinom(300, 1, 0.4),
  think   = rbinom(300, 1, 0.3),
  n_events = rpois(300, 20),
  actor   = rep(letters[1:10], each = 30)
)
d$success <- rbinom(300, 1, plogis(-0.5 + 0.8 * d$hint))
fit <- outcome_model(d, outcome = "success",
  predictors = c("hint", "think"),
  adjust = "n_events")
effects_table(fit)

```

passage_time	<i>Mean First Passage Times</i>
--------------	---------------------------------

Description

Computes the full matrix of mean first passage times (MFPT) for a Markov chain. Element M_{ij} is the expected number of steps to travel from state i to state j for the first time. The diagonal equals the mean recurrence time $1/\pi_i$.

Usage

```

passage_time(x, states = NULL, normalize = TRUE)

## S3 method for class 'net_mpt'
print(x, digits = 1, ...)

## S3 method for class 'net_mpt_group'
print(x, ...)

## S3 method for class 'net_mpt'
summary(object, ...)

## S3 method for class 'summary.net_mpt'
print(x, ...)

## S3 method for class 'net_mpt'
plot(
  x,
  log_scale = TRUE,
  digits = 1,
  title = "Mean First Passage Times",
  low = "#004d00",
  high = "#ccffcc",
  ...
)

```

Arguments

<code>x</code>	A netobject, cograph_network, tna object, row-stochastic numeric transition matrix, or a wide sequence data.frame (rows = actors, columns = time-steps; a relative transition network is built automatically). For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_mpt</code> or <code>net_mpt_group</code> (or its <code>summary()</code>).
<code>states</code>	Character vector. Restrict output to these states. NULL (default) keeps all states.
<code>normalize</code>	Logical. If TRUE (default), rows that do not sum to 1 are normalized automatically (with a warning).
<code>digits</code>	Integer. Decimal places displayed in cells. Default 1.
<code>...</code>	Ignored. In <code>plot.net_mpt()</code> , <code>print.net_mpt()</code> , <code>print.summary.net_mpt()</code> and <code>summary.net_mpt()</code> : Ignored. In <code>print.net_mpt_group()</code> : Forwarded to <code>print.net_mpt</code> for each element.
<code>object</code>	A <code>net_mpt</code> object (for summary).
<code>log_scale</code>	Logical. Apply log transform to the fill scale for better contrast? Default TRUE.
<code>title</code>	Character. Plot title.
<code>low</code>	Character. Hex colour for the low end (short passage time). Default dark green "#004d00".
<code>high</code>	Character. Hex colour for the high end (long passage time). Default pale green "#ccffcc".

Details

Uses the Kemeny-Snell fundamental matrix formula:

$$M_{ij} = \frac{Z_{jj} - Z_{ij}}{\pi_j}, \quad Z = (I - P + \Pi)^{-1}$$

where $\Pi_{ij} = \pi_j$. Requires an ergodic (irreducible, aperiodic) chain.

Value

An object of class "net_mpt" with:

matrix Full $n \times n$ MFPT matrix. Row i , column j = expected steps from state i to state j . Diagonal = mean recurrence time $1/\pi_i$.

stationary Named numeric vector: stationary distribution π .

return_times Named numeric vector: $1/\pi_i$ per state.

states Character vector of state names.

For a `netobject_group` the result is a "net_mpt_group": a named list holding one `net_mpt` per group.

In `print.net_mpt()`: x, invisibly.

In `print.net_mpt_group()`: x invisibly.

In `summary.net_mpt()`: `summary.net_mpt` returns an object of class "summary.net_mpt": a list whose table is a data frame with one row per state and columns state, return_time, stationary,

mean_out (mean steps to other states) and mean_in (mean steps from other states), and whose object is the net_mpt it summarises. Its print method shows the table.

In `plot.net_mpt()`: `plot.net_mpt` returns a ggplot object: a from-by-to heatmap of the mean first passage time matrix.

In `print.summary.net_mpt()`: x, invisibly.

References

Kemeny, J.G. and Snell, J.L. (1976). *Finite Markov Chains*. Springer-Verlag.

See Also

[markov_stability](#), [build_network](#)

Examples

```
net <- build_network(as.data.frame(trjectories), method = "relative")
pt <- passage_time(net)
print(pt)

plot(pt)
```

pathways

Extract Pathways from Higher-Order Network Objects

Description

Extracts higher-order pathway strings suitable for `cograph::plot_simplicial()`. Each pathway represents a multi-step dependency: source states lead to a target state.

For `net_hon`: extracts edges where the source node is higher-order (`order > 1`), i.e., the transitions that differ from first-order Markov.

For `net_hypa`: extracts anomalous paths (over- or under-represented relative to the hypergeometric null model).

For `net_mogen`: extracts all transitions at the optimal order (or a specified order).

Usage

```
pathways(x, ...)
```

S3 method for class 'net_hon'

```
pathways(x, min_count = 1L, min_prob = 0, top = NULL, order = NULL, ...)
```

S3 method for class 'net_hypa'

```
pathways(x, type = "all", ...)
```

```
## S3 method for class 'netobject'
pathways(x, ho_method = c("hon", "hypo"), ...)

## S3 method for class 'net_association_rules'
pathways(x, top = NULL, min_lift = NULL, min_confidence = NULL, ...)

## S3 method for class 'net_link_prediction'
pathways(x, method = NULL, top = 10L, evidence = TRUE, max_evidence = 3L, ...)

## S3 method for class 'net_mogen'
pathways(x, order = NULL, min_count = 1L, min_prob = 0, top = NULL, ...)
```

Arguments

x	A higher-order network object (net_hon, net_hypo, net_mogen), a netobject, a net_association_rules or a net_link_prediction.
...	Additional arguments passed on to the method.
min_count	Integer. Minimum transition count to include (default: 1). Filters noise from rare observations. Used by the net_hon and net_mogen methods.
min_prob	Numeric. Minimum transition probability to include (default: 0). Useful for filtering weak transitions. Used by the net_hon and net_mogen methods.
top	Integer or NULL. Keep only the top N pathways: ranked by count for net_hon and net_mogen, by lift then confidence for net_association_rules, and by score for net_link_prediction. Default NULL (all) everywhere except net_link_prediction, whose default is 10.
order	Integer or NULL. For net_hon, keep only pathways whose source is of this order; default NULL = every order above 1. For net_mogen, the Markov order to extract; default NULL = the optimal order from model selection.
type	Character. Which anomalies to include: "all" (default), "over", or "under".
ho_method	Character. Higher-order method: "hon" (default) or "hypo".
min_lift	Numeric or NULL. Additional lift filter applied on top of the object's original threshold (default: NULL).
min_confidence	Numeric or NULL. Additional confidence filter (default: NULL).
method	Character or NULL. Which prediction method to use. Default: first method in the object.
evidence	Logical. If TRUE, include common neighbor evidence nodes in each pathway. Default: TRUE.
max_evidence	Integer. Maximum number of evidence nodes per pathway (default: 3).

Value

A character vector of pathway strings in arrow notation (e.g. "A B -> C"), suitable for `cograph::plot_simplicial()`. Every method returns this shape, and `character(0)` when nothing survives its filters.

Methods (by class)

- `pathways(net_hon)`: Extract higher-order pathways from HON
- `pathways(net_hypa)`: Extract anomalous pathways from HYPA
- `pathways(netobject)`: Extract pathways from a `netobject`
Builds a Higher-Order Network (HON) from the `netobject`'s sequence data and returns the higher-order pathways. Requires that the `netobject` was built from sequence data (has `$data`).
- `pathways(net_association_rules)`: Extract pathways from association rules
Converts association rules $\{A, B\} \Rightarrow \{C\}$ into pathway strings "A B -> C" suitable for `cograph::plot_simplicial()`. Antecedent items become source nodes; consequent items become the target. Rules whose antecedent and consequent share the same item set describe the same simplex, so only the highest-lift rule per item set is returned; top is applied after that de-duplication.
- `pathways(net_link_prediction)`: Extract pathways from link predictions
Converts predicted links into pathway strings for `cograph::plot_simplicial()`. When `evidence = TRUE` (default), each predicted edge $A \rightarrow B$ is enriched with common neighbors that structurally support the prediction, producing "A cn1 cn2 -> B".
- `pathways(net_mogen)`: Extract transition pathways from MOGen

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"))
hon <- build_hon(seqs, max_order = 3)
pw <- pathways(hon)

trans <- list(c("A","B","C"), c("A","B"), c("B","C","D"), c("A","C","D"))
rules <- association_rules(trans, min_support = 0.3, min_confidence = 0.3,
                           min_lift = 0)
pathways(rules)

seqs <- data.frame(
  V1 = sample(LETTERS[1:5], 50, TRUE),
  V2 = sample(LETTERS[1:5], 50, TRUE),
  V3 = sample(LETTERS[1:5], 50, TRUE)
)
net <- build_network(seqs, method = "relative")
pred <- predict_links(net, methods = "common_neighbors")
pathways(pred)
```

path_counts

*Count Path Frequencies in Trajectory Data***Description**

Counts the frequency of k -step paths (k -grams) across all trajectories. Useful for understanding which sequences dominate the data before applying formal models.

Usage

```
path_counts(data, k = 2L, top = NULL)
```

Arguments

data	A list of character vectors (trajectories) or a data.frame (rows = trajectories, columns = time points).
k	Integer >= 2. Length of the path / n-gram (default 2). A k of 2 counts individual transitions; k of 3 counts two-step paths, etc. Must be a whole number; a non-integer value is an error rather than being silently truncated.
top	Integer or NULL. If set, returns only the top N most frequent paths (default NULL = all).

Value

A data frame with one row per distinct k-gram, sorted by count (descending), with columns path (the k states in arrow notation, e.g. "A -> B"), count and proportion (share of all k-grams, rounded to 4 decimal places).

Examples

```
trajs <- list(c("A","B","C","D"), c("A","B","D","C"))
path_counts(trajs, k = 2)

path_counts(trajs, k = 3, top = 10)
```

path_dependence	<i>Per-Context Path Dependence at Order k</i>
-----------------	---

Description

Diagnoses where a chain’s order-1 Markov assumption fails by comparing, for each order-k context $(s_1, \dots, s_{k-1})$, the empirical next-state distribution $P(s_k \mid s_1, \dots, s_{k-1})$ against the order-1 prediction $P(s_k \mid s_{k-1})$ that uses only the most recent state. Returns a tidy per-context table sorted by Kullback-Leibler divergence so the analyst can see exactly *which* histories carry extra predictive information.

Usage

```
path_dependence(x, order = 2L, min_count = 5L, base = 2)

## S3 method for class 'net_path_dependence'
print(x, top = 10L, digits = 3L, ...)
```

```
## S3 method for class 'net_path_dependence'
summary(object, ...)

## S3 method for class 'summary.net_path_dependence'
print(x, digits = 3L, ...)

## S3 method for class 'net_path_dependence'
plot(x, top = 15L, title = NULL, ...)
```

Arguments

x	A wide sequence data.frame / matrix (rows = actors, columns = time-steps), or a netobject that carries the source data. For the print() and plot() methods: an object of class net_path_dependence (or its summary()).
order	Integer. Order of the conditioning context. order = 2 (default) compares 2-step memory against 1-step; order = 3 compares 3-step memory; etc. Must be a whole number; a non-integer value is an error rather than being silently truncated.
min_count	Integer. Drop contexts seen fewer than this many times. Default 5. Very small samples produce noisy KL estimates.
base	Numeric. Logarithm base for entropy and KL. Default 2 (bits).
top	In print.net_path_dependence(): Integer. Number of top contexts to show. Default 10. In plot.net_path_dependence(): Integer. Number of contexts to show (top by KL). Default 15.
digits	Integer. Digits to round numeric output. Default 3.
...	In plot.net_path_dependence(), print.net_path_dependence(), print.summary.net_path_dependence() and summary.net_path_dependence(): Ignored.
object	For the summary() method: an object of class net_path_dependence.
title	Character or NULL. Plot title. Default NULL, which builds "Path dependence: order k vs order 1" from the fitted order.

Details

For each context $c = (s_1, \dots, s_{k-1})$ occurring at least min_count times, the function computes:

- the empirical conditional $P_k(\cdot | c)$ from k-gram counts;
- the order-1 prediction $P_1(\cdot | s_{k-1})$ from the most recent state alone (the bigram-marginal estimator);
- the entropy drop $H(P_1) - H(P_k)$ - bits of uncertainty removed by extending memory by one step in this specific context;
- the Kullback-Leibler divergence $D_{KL}(P_k \| P_1)$ - bits of "surprise" if you used the order-1 model when the order-k model is true.

KL = 0 means longer history adds no information for that context. H_drop > 0 means longer history sharpens the prediction; H_drop < 0 indicates the order-k context happens to spread probability across more outcomes than order-1 alone (small-sample noise or genuine context-induced uncertainty - inspect n).

Contexts where `flips = TRUE` are the substantively interesting ones: the longer history changes the *modal* prediction, not just its confidence.

Pair this with [markov_order_test](#) (which decides whether order-k is needed *globally*) to see the chain-level decision broken down per context.

Value

An object of class "net_path_dependence" with

contexts tidy data.frame, one row per order-k context, sorted by KL descending. Columns: context (e.g. "A -> B"), n (count), H_order1 (entropy of $P(\cdot \mid s_{k-1})$), H_orderk (entropy of $P(\cdot \mid \text{context})$), H_drop (= H_order1 - H_orderk), KL (= $D_{KL}(P_k \parallel P_1)$), top_o1 (most likely next state under order-1), top_ok (most likely next state under order-k), flips (logical: did the most likely next state change?).

chain list with chain-level summaries: KL_weighted (count-weighted mean KL across contexts), H_drop_weighted (count-weighted mean entropy drop), n_contexts, n_flips (contexts where the most-likely next state changed).

order integer

base numeric

min_count integer

states character vector

In `print.net_path_dependence()` and `print.summary.net_path_dependence()`: x invisibly.

In `summary.net_path_dependence()`: A `summary.net_path_dependence` with the full sorted table and chain-level summaries.

In `plot.net_path_dependence()`: A ggplot object.

Methods

- `plot.net_path_dependence()`: Lollipop chart of per-context KL divergence, sorted descending. Point size is proportional to context count; points where the modal next state flips between orders are marked with an X to highlight substantively meaningful order-2 effects.

References

Cover, T.M. & Thomas, J.A. (2006). *Elements of Information Theory*, 2nd ed., chapters 2 and 4. Wiley. (KL divergence and conditional entropy.)

See Also

[markov_order_test](#), [transition_entropy](#), [build_mogen](#)

Examples

```
data(trajectories, package = "Nestimate")
pd <- path_dependence(as.data.frame(trajectories), order = 2)
print(pd)
summary(pd)
```

```
plot(pd)
```

```
permutation
```

```
Permutation Test for Network Comparison
```

Description

Tests whether two networks estimated by `build_network` differ more than chance would produce. The sequences (or rows) of both networks are pooled, the group labels are shuffled `iter` times, both networks are re-estimated on every shuffle, and the observed differences are compared with the shuffled ones. Works with every built-in method and with registered custom estimators.

Usage

```
permutation(
  x,
  y = NULL,
  iter = 1000L,
  alpha = 0.05,
  paired = FALSE,
  adjust = "none",
  measures = NULL,
  nlambda = 50L,
  seed = NULL,
  actor = NULL
)

## S3 method for class 'net_permutation'
print(x, ...)

## S3 method for class 'net_permutation'
summary(object, ...)

## S3 method for class 'net_permutation_group'
print(x, ...)

## S3 method for class 'net_permutation_group'
summary(object, ...)

## S3 method for class 'wtna_perm_mixed'
print(x, ...)

## S3 method for class 'wtna_perm_mixed'
summary(object, ...)
```

Arguments

<code>x</code>	A netobject (from <code>build_network</code>) or a <code>net_edge_betweenness</code> object. For the <code>print()</code> method: an object of class <code>net_permutation</code> , <code>net_permutation_group</code> or <code>wtna_perm_mixed</code> .
<code>y</code>	A netobject (from <code>build_network</code>) or a <code>net_edge_betweenness</code> object. Must use the same method and have the same nodes as <code>x</code> . Default NULL: when <code>x</code> is a <code>netobject_group</code> (or an <code>mcm1</code>) and <code>y</code> is left NULL, every pair of groups is tested and the result is a <code>net_permutation_group</code> named "<group i> vs <group j>".
<code>iter</code>	Integer. Number of permutation iterations (default: 1000).
<code>alpha</code>	Numeric. Significance level (default: 0.05).
<code>paired</code>	Logical. If TRUE, permute within pairs (requires equal number of observations in <code>x</code> and <code>y</code>). Default: FALSE.
<code>adjust</code>	Character. p-value adjustment method passed to <code>p.adjust</code> (default: "none"). Common choices: "holm", "BH", "bonferroni".
<code>measures</code>	Character vector of centrality measures to permutation-test in addition to the edges, or "all" for every built-in measure. Default NULL (edges only). When supplied, the result gains a <code>\$centralities</code> block matching the layout of <code>tna::permutation_test(measures =)</code> : per state and measure it reports the observed difference, an effect size (difference / SD of the permutation null), and a permutation p-value, all using the same permuted networks as the edge test. Not supported for <code>net_edge_betweenness</code> inputs.
<code>nlambda</code>	Integer. Number of lambda values for the EBIC-glasso regularisation path (only used when <code>method = "glasso"</code>). Higher values give finer lambda resolution at the cost of speed. Default: 50.
<code>seed</code>	Integer or NULL. RNG seed for reproducibility.
<code>actor</code>	Character or NULL. Name of the column identifying the actor each sequence belongs to: the person whose sessions they are, or the team of a student. Looked up in the network's <code>\$metadata</code> (e.g. "student_id" when sessions are nested in students, "Group" for students nested in teams) or in its wide sequence data. When supplied, whole actors are permuted and the ICC and design effect are reported; see the sections <i>Nested data and actor</i> and <i>ICC and design effect</i> . Supported for transition methods ("relative", "frequency", "co_occurrence"); cannot be combined with <code>paired = TRUE</code> , which is the special case of one actor per pair. Default NULL: sequences are permuted individually.
<code>...</code>	In <code>print.net_permutation()</code> , <code>print.net_permutation_group()</code> , <code>print.wtna_perm_mixed()</code> , <code>summary.net_permutation()</code> , <code>summary.net_permutation_group()</code> and <code>summary.wtna_perm_mixed()</code> . Additional arguments (ignored).
<code>object</code>	For the <code>summary()</code> method: an object of class <code>net_permutation</code> , <code>net_permutation_group</code> or <code>wtna_perm_mixed</code> .

Value

An object of class "net_permutation" containing:

x The first netobject.

y The second netobject.

diff Observed difference matrix ($x - y$).

diff_sig Observed difference where $p < \alpha$, else 0.

p_values P-value matrix (adjusted if `adjust != "none"`).

effect_size Effect size matrix (observed diff / SD of permutation diffs).

summary Long-format data frame, one row per edge present in either network (undirected networks keep one row per unordered pair), with columns `from`, `to`, `weight_x`, `weight_y`, `diff`, `effect_size`, `p_value`, `sig`.

global Data frame of the two NCT-style global statistics, one row each: `statistic` ("M", the sum of absolute edge differences, and "S", the largest absolute edge difference), `observed`, and `p_value` from the same permutation null as the edge test. Absent on the edge-betweenness path.

method The network estimation method.

source_method For edge-betweenness tests, the source network method.

iter Number of permutation iterations.

alpha Significance level used.

paired Whether paired permutation was used.

adjust p-value adjustment method used.

actor The actor column name, or NULL.

n_actors Number of distinct actors, or NULL.

null_sd Matrix of the SD of each edge difference over the permutation null (the effect-size denominator).

null_sd_m SD of the global M statistic over the permutation null. Absent on the edge-betweenness path.

clustering Present only with actor. One-row data frame: `n_sequences`, `n_actors`, `design` ("between", "within", "mixed"), `icc` with `icc_ci_lower`/`icc_ci_upper` (how alike sequences of one actor are; see [permutation_diagnostics](#)), `deff_edges` (median over edges) and `deff_global` (for M): the actor-level over the sequence-level null variance, drawn in the same run (the design effect; Kish, 1965). `min_p` is the smallest attainable p-value.

clustering_edges Present only with actor. One row per edge of summary: `from`, `to`, `icc`, `null_sd_actor`, `null_sd_sequence`, `deff`.

centralities Present only when `measures` is supplied. A list with stats (one row per state-by-measure: `state`, `centrality`, `diff_true`, `effect_size`, `p_value`), `diffs_true` (wide observed differences), and `diffs_sig` (observed differences where $p < \alpha$, else 0).

Grouped input returns a "net_permutation_group" (a named list of net_permutation results): one element per matching group name when both `x` and `y` are netobject_groups, or one per group pair when `y` is NULL. Two `wtna_mixed` inputs return a "wtna_perm_mixed" with `$transition` and `$cooccurrence` results.

In `print.net_permutation()` and `print.wtna_perm_mixed()`: The input object, invisibly.

In `summary.net_permutation()`: The `$summary` data frame: one row per edge present in either network, with columns `from`, `to`, `weight_x`, `weight_y`, `diff`, `effect_size`, `p_value`, `sig`.

In `print.net_permutation_group()`: x invisibly.

In `summary.net_permutation_group()`: The per-group summaries stacked into one data frame: the columns of `summary.net_permutation` prefixed by a group column naming the group (or group pair) each row came from.

In `summary.wtna_perm_mixed()`: A list with transition and co-occurrence permutation summaries.

What is tested

Two kinds of question are answered from the same shuffles.

Edge tests One test per edge: is the difference in this edge's weight, $x - y$, larger than the shuffles produce? Reported in `summary()` with an effect size (observed difference divided by the SD of the shuffled differences) and a p-value (number of shuffles at least as extreme + 1) / (iter + 1). With many edges, some fall below alpha by chance; use `adjust` to correct for that.

Global test One test for the whole network: do the two networks differ at all? Two statistics, as in the Network Comparison Test (van Borkulo et al., 2023): **M**, the sum of the absolute edge differences (the total amount of difference), and **S**, the largest absolute edge difference. Being a single test, it needs no multiplicity correction. It is shown by `print()`.

The smallest attainable p-value is $1 / (\text{iter} + 1)$; with the default `iter = 1000` it is 0.000999, meaning no shuffle came close.

Nested data and actor

Shuffling treats every sequence as an exchangeable unit. When several sequences come from the same actor (sessions of one person, students of one team), actor names the column identifying that actor. The shuffle then respects it (Good, 2005; Anderson & ter Braak, 2003): a person whose sequences are all in one group moves to the other group as a whole; a person with sequences in both groups has their labels shuffled among their own sequences only. Mixed designs combine the two. The observed differences do not change; only the p-values and effect sizes do. With few persons there are few distinct ways to shuffle them, and a warning (class `nestimate_few_actors`) is raised when no p-value could fall below alpha.

actor is available for transition networks ("relative", "frequency", "co_occurrence"). Association networks ("cor", "pcor", "glasso", ...) do not keep row identifiers after estimation and raise `nestimate_actor_unsupported`.

ICC and design effect

With actor, `print()` also reports:

ICC The intraclass correlation, the proportion of the total variance that lies between actors (Shrout & Fleiss, 1979). An ICC close to 0 indicates little evidence of a nesting effect. Computed as the one-way ANOVA ICC of each sequence's transition shares within each group, averaged over edges weighted by edge frequency, jackknife bias-corrected, with a 95% interval from the leave-one-actor-out jackknife (Efron & Tibshirani, 1993).

Design effect The ratio of the variance of an estimate under the clustered design to its variance had the units been sampled independently (Kish, 1965). Here: the variance of the shuffled differences when whole actors are moved, divided by the variance when single sequences are

moved, both drawn in the same run. Reported as the median over edges and for the global statistic M . For equal numbers of sequences per actor m , Kish gives the approximation $1 + (m - 1) * ICC$.

Reading the printed output

```
Permutation Test: Transition Network (relative probabilities) [directed]
Iterations: 1000 | Alpha: 0.05 | Actor: Group (200 actors)
Nodes: 9 | Edges tested: 78 | Significant: 42
Global test (networks differ overall?): M = 2.612 (p = 0.000999) | ...
Nesting in Group: ICC = -0.002 [95% CI -0.006, 0.002] | between design
Design effect (1 = nesting does not matter): edges 1.03 | global 1.20
```

Line 2: settings, and the actor column with its number of actors. Line 3: edges present in either network and how many differ at alpha. Line 4: the global test. Lines 5-6, only with actor: the ICC with its interval, whether actors sit in one group (between), in both (within) or either (mixed), and the design effects.

Other inputs

For transition methods, per-sequence count matrices are computed once and each shuffle only re-sums them, which keeps large iter fast. For association methods the estimator is re-run on every shuffle. If a transition network rests on a single sequence, a warning (class `nestimate_single_sequence`) says it cannot be validated by resampling.

`permutation()` also accepts two `net_edge_betweenness` objects. It then permutes the source networks, recomputes edge betweenness for each shuffle, and tests the edge-betweenness differences. Both objects must come from the same source method and use the same `invert` setting.

Methods

- `summary.net_permutation_group()`: Returns a combined summary data frame across all groups.

References

- Anderson, M. J., & ter Braak, C. J. F. (2003). Permutation tests for multi-factorial analysis of variance. *Journal of Statistical Computation and Simulation*, 73(2), 85-113.
- Efron, B., & Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman & Hall.
- Good, P. (2005). *Permutation, Parametric, and Bootstrap Tests of Hypotheses* (3rd ed.). Springer.
- Kish, L. (1965). *Survey Sampling*. Wiley.
- Shrout, P. E., & Fleiss, J. L. (1979). Intraclass correlations: Uses in assessing rater reliability. *Psychological Bulletin*, 86(2), 420-428.
- van Borkulo, C. D., van Bork, R., Boschloo, L., Kossakowski, J. J., Tio, P., Schoevers, R. A., Borsboom, D., & Waldorp, L. J. (2023). Comparing network structures on three aspects: A permutation test. *Psychological Methods*, 28(6), 1273-1285.

See Also

[permutation_diagnostics](#) to compare the actor-level and ordinary tests side by side; [bayes_compare](#) for the Bayesian complement: instead of "is this difference more extreme than chance?" it answers "how probable is a difference, and how large?"; [build_network](#), [bootstrap_network](#), [print.net_permutation](#), [summary.net_permutation](#)

Examples

```
s1 <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
s2 <- data.frame(V1 = c("A", "C", "B"), V2 = c("C", "B", "A"))
n1 <- build_network(s1, method = "relative")
n2 <- build_network(s2, method = "relative")
perm <- permutation(n1, n2, iter = 10)
```

```
set.seed(1)
d1 <- data.frame(V1 = sample(LETTERS[1:4], 20, TRUE),
                 V2 = sample(LETTERS[1:4], 20, TRUE),
                 V3 = sample(LETTERS[1:4], 20, TRUE))
d2 <- data.frame(V1 = sample(LETTERS[1:4], 20, TRUE),
                 V2 = sample(LETTERS[1:4], 20, TRUE),
                 V3 = sample(LETTERS[1:4], 20, TRUE))
net1 <- build_network(d1, method = "relative")
net2 <- build_network(d2, method = "relative")
perm <- permutation(net1, net2, iter = 100, seed = 42)
print(perm)
summary(perm)
```

```
# Students are nested in teams, and Achiever is a team-level label:
# permute whole teams, not single students
net <- build_network(group_regulation_long, method = "relative",
                    actor = "Actor", action = "Action", time = "Time",
                    group = "Achiever")
permutation(net, iter = 100, actor = "Group", seed = 1)
```

permutation_diagnostics

Does Nesting Bias a Permutation Test?

Description

Shows what treating nested sequences as independent would cost. The comparison is run twice on the same data: once with the ordinary [permutation](#) test, which shuffles single sequences, and once with actor, which shuffles whole actors (the persons whose sessions they are, or the teams of students). The result places the two side by side, with the ICC and design effect that explain any difference between them. See the sections *Nested data and actor* and *ICC and design effect* of [permutation](#) for what these quantities mean.

Usage

```
permutation_diagnostics(
  x,
  y = NULL,
  actor,
  iter = 1000L,
  alpha = 0.05,
  level = c("overall", "edges"),
  seed = NULL
)
```

Arguments

x	A <code>netobject_group</code> (every pair of groups is diagnosed) or a <code>netobject</code> (then <code>y</code> is required). Transition methods only (" <code>relative</code> ", " <code>frequency</code> ", " <code>co_occurrence</code> ").
y	A <code>netobject</code> to compare with <code>x</code> , or <code>NULL</code> .
actor	Character. Column identifying the actor each sequence belongs to, as in permutation .
iter	Integer. Permutation iterations for each of the two tests. Default 1000.
alpha	Numeric. Significance level. Default 0.05.
level	Character. " <code>overall</code> " (default): one row per group pair. " <code>edges</code> ": one row per edge per pair.
seed	Integer or <code>NULL</code> . RNG seed; both tests use the same seed.

Value

A `data.frame`.

With `level = "overall"`, one row per compared pair:

pair "<x> vs <y>".

n_sequences, n_actors Sequences and distinct actors in the pair.

design "`between`" (every actor in one group), "`within`" (every actor in both groups) or "`mixed`".

icc, icc_ci_lower, icc_ci_upper How alike the sequences of one actor are, with a 95% interval; as printed by [permutation](#). NA interval with fewer than 3 actors.

deff_edges Median over edges of the design effect, the ratio of the actor-level to the sequence-level null variance of the edge difference (Kish, 1965).

deff_global The same ratio for the global M statistic.

p_global_sequence, p_global_actor Permutation p-values of M when sequences or whole actors are reassigned.

sig_edges_sequence, sig_edges_actor Edges with $p < \alpha$ under each test.

edges_changed Edges significant under one test but not the other.

min_p_actor Smallest p-value an exact actor-level test can produce, $\max(1 / \text{arrangements}, 1 / (\text{iter} + 1))$.

persistence_landscape *Persistence Landscape*

Description

Computes the persistence landscape (Bubenik 2015) from a persistence diagram. Each (birth, death) pair contributes a tent function

$$\Lambda_{(b,d)}(t) = \max(0, \min(t - b, d - t)).$$

The k -th landscape function $\lambda^{(k)}(t)$ is the k -th largest of $\{\Lambda_{(b_i,d_i)}(t)\}_i$ at each t . Landscapes are stable under bottleneck distance and form a Banach-space embedding of persistence diagrams.

Essential classes are excluded: a tent function is undefined for an infinitely-lived class on a finite grid, so pairs with death = Inf (VR mode) and pairs with death = 0 but birth > 0 (the clique-mode encoding of an essential class) are dropped before the landscape is built. When no finite pair remains in the requested dimension, every landscape function is zero on the grid.

Usage

```
persistence_landscape(ph, k_max = 5L, dimension = 1L, t_grid = NULL)

## S3 method for class 'persistence_landscape'
print(x, ...)

## S3 method for class 'persistence_landscape'
plot(x, ...)
```

Arguments

ph	A persistent_homology object or a data.frame with columns dimension, birth, death.
k_max	Maximum landscape index to compute (default 5). Must be a single positive integer.
dimension	Integer scalar – which homology dimension to compute the landscape for. Default 1.
t_grid	Numeric vector of evaluation points. NULL (default) uses an even grid of 200 points covering the union of pair intervals.
x	For the print() and plot() methods: an object of class persistence_landscape.
...	In plot.persistence_landscape() and print.persistence_landscape(): Ignored.

Value

A persistence_landscape object with:

landscape Data frame: k, t, value.

dimension Integer scalar.

k_max Integer scalar.

t_grid Numeric vector.

In `print.persistence_landscape()`: The input, invisibly.

In `plot.persistence_landscape()`: A ggplot.

References

Bubenik, P. (2015). Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research* **16**, 77-102.

Examples

```
mat <- matrix(c(0, .6, .5, .6, 0, .4, .5, .4, 0), 3, 3)
rownames(mat) <- colnames(mat) <- c("A", "B", "C")
ph <- persistent_homology(mat, n_steps = 5)
pl <- persistence_landscape(ph, k_max = 3, dimension = 0)
```

persistent_homology *Persistent Homology*

Description

Computes persistent homology via full boundary-matrix reduction over $\mathbb{Z}/2$ (Edelsbrunner, Letscher & Zomorodian 2000). The returned persistence diagram pairs each k -dimensional homology class to the simplex whose addition creates it (birth) and the simplex whose addition destroys it (death). Essential classes - those never killed - are reported with death = $\emptyset$ in clique mode (similarity scale, descending) and death = Inf in VR mode (distance scale, ascending).

Two filtration modes are supported:

`type = "clique"` Weighted clique filtration. Input is treated as a similarity matrix; high-weight simplices appear early. For each k -simplex σ , the filtration value is $\min_{(i,j) \in \sigma} |w(i,j)|$. Thresholds run high to low.

`type = "vr"` Vietoris-Rips filtration on a non-negative distance matrix. For each k -simplex σ , the filtration value is $\max_{(i,j) \in \sigma} d(i,j)$. Thresholds run low to high. Use `max_scale` to cap the filtration diameter.

Usage

```
persistent_homology(
  x,
  n_steps = 20L,
  max_dim = 3L,
  type = "clique",
  max_scale = NULL
```

```
)

## S3 method for class 'persistent_homology'
print(x, ...)

## S3 method for class 'persistent_homology'
plot(x, combined = TRUE, ...)
```

Arguments

<code>x</code>	A square matrix, tna, or netobject. For type = "vr", must be a non-negative distance matrix. A simplicial_complex carrying a \$filtration vector (as returned by build_simplicial(type = "vr")) is also accepted and is used directly: its stored filtration values are reduced as they are, so type is taken from the complex rather than this argument. For the print() and plot() methods: an object of class persistent_homology.
<code>n_steps</code>	Number of grid points for the reported Betti curve (default 20). The persistence diagram itself is exact - it does not depend on n_steps.
<code>max_dim</code>	Maximum simplex dimension to track (default 3).
<code>type</code>	Filtration: "clique" (default, similarity-weighted) or "vr" (Vietoris-Rips on distances).
<code>max_scale</code>	For type = "vr" only: cap on edge length. Edges with $d(i, j) > \text{max_scale}$ are excluded. NULL (default) uses $\max(d)$.
<code>...</code>	In plot.persistent_homology(): Ignored. In print.persistent_homology(): Additional arguments (unused).
<code>combined</code>	When TRUE (default), the two panels are stitched side-by-side via <code>gridExtra::arrangeGrob</code> . When FALSE, returns a named list (betti_curve, persistence) of ggplots.

Value

A persistent_homology object with:

betti_curve Data frame: threshold, dimension, betti.

persistence Data frame of birth-death pairs, one row per homology class: dimension, birth, death, persistence. Sorted by descending persistence. Essential classes are included, with death = 0 and persistence = birth in clique mode, and death = Inf, persistence = Inf in VR mode (the plot method caps those for display only).

thresholds Numeric vector of grid thresholds.

mode Either "clique" or "vr".

In print.persistent_homology(): The input object, invisibly.

In plot.persistent_homology(): A grid grob (invisibly) when combined = TRUE; a named list of two ggplots when combined = FALSE.

Methods

- `plot.persistent_homology()`: Two panels: Betti curve (threshold vs Betti number) and persistence diagram (birth vs death). Persistence pairs come from full boundary- matrix reduction; essential classes are shown at the filtration boundary (death = 0 in clique mode; in VR mode their stored death = Inf is capped for display at the largest finite value in the diagram or on the threshold grid, so they still render).

References

Edelsbrunner, H., Letscher, D., & Zomorodian, A. (2000). Topological persistence and simplification. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, 454-463. Journal version: *Discrete & Computational Geometry* (2002) **28**, 511-533.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
ph <- persistent_homology(mat, n_steps = 10)
print(ph)
```

plot_mosaic

Draw a Marimekko / Mosaic Plot from a Tidy Data Frame

Description

Low-level rectangle-coordinate builder for marimekko (mosaic) plots. Column widths are proportional to the per-column total of weight; within each column, segments stack to height 1 with sub-heights proportional to each row's share of that column's total.

Usage

```
plot_mosaic(
  data,
  x,
  y,
  weight,
  fill = "y",
  colors = NULL,
  show_labels = TRUE,
  label_size = 3.5,
  x_label = NULL,
  y_label = NULL
)
```

Arguments

data	A data.frame in long form. Must contain the columns named in x, y, and weight.
x	Column name for the X (column) variable.
y	Column name for the Y (segment) variable.
weight	Column name for the cell weight (e.g. count).
fill	Either "y" (color by Y category, e.g. state – default) or the name of another column to map to fill (e.g. a residual column for diverging color).
colors	Optional fill colors. Either an unnamed vector applied in level order (when fill = "y", length must be at least the number of distinct y levels), or a named lookup (c(plan = "#0072B2")) overriding only the levels you name. Defaults to recycled Okabe-Ito.
show_labels	If TRUE, draw within-segment percentage labels.
label_size	Numeric size for segment labels.
x_label, y_label	Optional axis labels.

Details

Used internally by [plot_state_frequencies](#); exposed so that other plot methods (e.g. permutation-residual visualisations) can reuse the same geometry by supplying a different fill column.

Value

A ggplot object.

Examples

```
df <- data.frame(
  group = rep(c("A", "B", "C"), each = 3),
  state = rep(c("s1", "s2", "s3"), 3),
  count = c(10, 5, 2, 7, 8, 3, 4, 6, 12)
)
plot_mosaic(df, x = "group", y = "state", weight = "count")
```

plot_state_frequencies

Plot State Frequency Distributions

Description

Visualise state (node) frequency distributions across groups for any Nestimate object that carries sequence data: a single netobject, a netobject_group, an mcml model, or an htna network.

Usage

```
plot_state_frequencies(x, ...)

## S3 method for class 'nestimate_facet_plot'
print(x, ...)

## S3 method for class 'nestimate_facet_list'
print(x, ...)

## S3 method for class 'netobject'
plot_state_frequencies(
  x,
  style = "marimekko",
  metric = "prop",
  label = "prop",
  legend = "auto",
  legend_dir = "auto",
  legend_frame = "none",
  sort_states = "frequency",
  colors = NULL,
  label_size = 3.5,
  abbreviate = FALSE,
  include_macro = FALSE,
  combine = "auto",
  ncol = NULL,
  node_groups = NULL,
  ...
)

## S3 method for class 'htna'
plot_state_frequencies(
  x,
  style = "marimekko",
  metric = "prop",
  label = "prop",
  legend = "auto",
  legend_dir = "auto",
  legend_frame = "none",
  sort_states = "frequency",
  colors = NULL,
  label_size = 3.5,
  abbreviate = FALSE,
  include_macro = FALSE,
  combine = "auto",
  ncol = NULL,
  node_groups = NULL,
  ...
)
```

```

## S3 method for class 'mcml'
plot_state_frequencies(
  x,
  style = "marimekko",
  metric = "prop",
  label = "prop",
  legend = "auto",
  legend_dir = "auto",
  legend_frame = "none",
  sort_states = "frequency",
  colors = NULL,
  label_size = 3.5,
  abbreviate = FALSE,
  include_macro = FALSE,
  combine = "auto",
  ncol = NULL,
  node_groups = NULL,
  ...
)

## S3 method for class 'netobject_group'
plot_state_frequencies(
  x,
  style = "marimekko",
  metric = "prop",
  label = "prop",
  legend = "auto",
  legend_dir = "auto",
  legend_frame = "none",
  sort_states = "frequency",
  colors = NULL,
  label_size = 3.5,
  abbreviate = FALSE,
  include_macro = FALSE,
  combine = "auto",
  ncol = NULL,
  node_groups = NULL,
  ...
)

## Default S3 method:
plot_state_frequencies(x, ...)

## S3 method for class 'state_freq'
print(x, digits = 1, max_states = 20L, ...)

## S3 method for class 'state_freq'

```

```
plot(x, ...)

## S3 method for class 'state_freq'
as.data.frame(x, ...)
```

Arguments

<code>x</code>	A netobject, netobject_group, mcm1, or htna object. For the <code>print()</code> , <code>plot()</code> and <code>as.data.frame()</code> methods: the <code>state_freq</code> object returned by <code>plot_state_frequencies()</code> ; for the <code>print()</code> methods of the per-facet figure, an object of class <code>nestimate_facet_plot</code> or <code>nestimate_facet_list</code> .
<code>...</code>	Reserved for future use. In <code>as.data.frame.state_freq()</code> , <code>plot.state_freq()</code> , <code>print.state_freq()</code> , <code>print.nestimate_facet_list()</code> and <code>print.nestimate_facet_plot()</code> : ignored.
<code>style</code>	One of: • "marimekko" (default) – per-group treemap panels with cumulative-width geometry; tile area = within-group state share. • "bars" – horizontal bars sorted by frequency, faceted per group. For chi-square mosaics of a (group x state) contingency table, use mosaic_plot directly – it is kept as a separate function with its own dispatch surface.
<code>metric</code>	For style = "bars": which value the bar length encodes – "prop" (default) or "freq". Treemap and hierarchical-marimekko areas always encode proportion within group.
<code>label</code>	Inline tile / bar annotation. All formats render on a single line. • "prop" (default) – proportion only, e.g. "66%" • "freq" – count only, e.g. "1,234" • "both" – count + proportion, e.g. "1,234 (66%)" • "state" – state name only, e.g. "Average" • "all" – state + proportion, e.g. "Average (66%)" • "none" – no inline labels
<code>legend</code>	Legend position. "auto" (default) resolves per style: "none" for style = "bars" (the y-axis already names every state, so a colour legend is redundant); "per_facet" for htna/mcm1 treemaps (state vocabularies differ per panel, so each gets its own legend); "bottom" for single-network and netobject_group treemaps (shared state vocabulary, one shared legend). Override with any of "bottom", "top", "right", "left", "none", or "per_facet". "per_facet" is silently demoted to "bottom" when every group shares the same state vocabulary (repeating one legend per panel would be redundant); when it does take effect it returns a gtable (requiring the gridExtra package) or a list of ggplots, per combine.
<code>legend_dir</code>	Legend internal layout: "auto" (default – horizontal for top/bottom, vertical for left/right), or force "horizontal" or "vertical" regardless of position.
<code>legend_frame</code>	"none" (default) for an unframed legend, or "border" to draw a thin grey rectangle around the legend ("legend enclosed in a square").
<code>sort_states</code>	One of "frequency" (default – most frequent first), "alpha", or "none".

colors	Optional colors overriding the default Okabe-Ito state palette. Either an unnamed vector applied in state order (length at least the number of unique states), or a named lookup (<code>c(plan = "#0072B2")</code>) overriding only the states you name.
label_size	Numeric size of inline labels (max size when ggfittext is installed – text auto-shrinks per tile).
abbreviate	Abbreviate state names. FALSE (default) shows full names; TRUE truncates to the first 3 characters via <code>base::abbreviate()</code> (which extends the truncation as needed to keep names unique after collision); a positive integer sets the target minimum length explicitly (e.g. <code>abbreviate = 4</code>). Affects tile labels, the legend, and the tidy table returned by <code>as.data.frame()</code> .
include_macro	For <code>mcm1</code> only: prepend a "macro" reference column showing aggregate state frequencies across all clusters. Default FALSE.
combine	For <code>legend = "per_facet"</code> only. "auto" (default) returns a single combined gtable for 1-3 panels and a list of ggplots (one per panel) for 4+ panels – many-cluster <code>mcm1</code> layouts read better as separate figures than as a tile grid. TRUE forces a combined gtable via gridExtra ; FALSE forces a list (knitr renders each at the chunk's full <code>fig.width / fig.height</code>).
ncol	For <code>legend = "per_facet"</code> with <code>combine = TRUE</code> : number of columns in the grid arrangement. NULL (default) picks 1, 2, or 3 columns based on the number of panels.
node_groups	Optional named character vector mapping node labels to semantic groups. When supplied, panels (or bars) are coloured / annotated by group rather than by individual state, so state-level palettes can collapse onto a smaller categorical legend.
digits	Number of decimal places for proportion / share columns. Default 1.
max_states	Cap on rows shown per group in the per-state table (default 20); the surplus is folded into a single "(+k more)" row. The full, uncapped table is returned by <code>as.data.frame(x)</code> .

Details

The `marimekko` layout is dispatched per class:

- For `mcm1`, where states partition cleanly into clusters, the chart is a hierarchical 2D `marimekko`: cluster columns of width proportional to cluster total, segments stacked vertically with heights proportional to within-cluster state proportions.
- For all other classes (`netobject`, `netobject_group`, `htna`), each group is rendered as its own panel containing a squarified treemap: each state becomes a rectangular tile whose AREA is exactly proportional to the state's share within that group. Single-panel when no groups exist; faceted when groups are present.

The bar style produces horizontal bars (state on the y-axis), faceted by group when groups exist. All variants use the Okabe-Ito palette.

Value

A `state_freq` object: a list with the rendered `$plot` (a `ggplot`; a `gtable` or a list of `ggplots` under `legend = "per_facet"`, per combine), the tidy `$table` (a `data.frame` with columns `group`, `state`, `count`, `proportion`, one row per (group, state) cell), and the call's `$style`, `$metric`, `$source_class`. The class supports `print()` (prints the tidy table and draws the chart), `plot()` (draws the chart alone), and `as.data.frame()` (returns the tidy table) – see the section below.

`print()` returns `x` invisibly (after printing the table and drawing the chart); `plot()` returns `invisible(NULL)` after drawing; `as.data.frame()` returns the tidy `data.frame`, one row per (group, state) cell with columns `group`, `state`, `count`, `proportion`.

The `state_freq` object

`plot_state_frequencies()` returns a `state_freq` object holding both the rendered chart and the tidy frequency table. `print()` shows the table in the console *and* draws the chart on the active graphics device, `plot()` draws the chart alone, and `as.data.frame()` returns the tidy table for downstream piping.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  data(group_regulation_long, package = "Nestimate")
  nw <- build_network(group_regulation_long,
                     method = "relative", format = "long",
                     actor = "Actor", action = "Action",
                     order = "Time", group = "Course")
  res <- plot_state_frequencies(nw)
  print(res)           # tidy frequency table in the console
  plot(res)            # ggplot chart
  head(as.data.frame(res))
}
```

predictability

Compute Node Predictability

Description

Computes the proportion of variance explained (R^2) for each node in the network, following Haslbeck & Waldorp (2018).

For `method = "glasso"` or `"pcor"`, predictability is computed analytically from the precision matrix:

$$R_j^2 = 1 - 1/\Omega_{jj}$$

where Ω is the precision (inverse correlation) matrix.

For `method = "cor"`, predictability is the multiple R^2 from regressing each node on its network neighbors (nodes with non-zero edges).

Usage

```

predictability(object, ...)

## S3 method for class 'netobject'
predictability(object, data = NULL, ...)

## S3 method for class 'netobject_ml'
predictability(object, ...)

## S3 method for class 'netobject_group'
predictability(object, ...)

```

Arguments

<code>object</code>	A <code>netobject</code> , <code>netobject_ml</code> , or <code>netobject_group</code> object.
<code>...</code>	Additional arguments (ignored).
<code>data</code>	Optional data frame of the original variables used to estimate the network. R^2 never needs it (it comes from the precision or correlation matrix stored on the object). It is used only for the RMSE column and defaults to <code>object\$data</code> ; when neither is available RMSE is NA.

Value

For `netobject`: a data frame with one row per node and columns `node` (character), `R2` (numeric, between 0 and 1) and `RMSE` (numeric, NA when no data is available).

For `netobject_ml`: a list with elements `$between` and `$within`, each such a data frame.

For `netobject_group`: a named list of such data frames, one per group.

A data frame with one row per node and columns `node`, `R2` and `RMSE`.

A list with `between` and `within` predictability data frames.

A named list of per-group predictability data frames.

References

Haslbeck, J. M. B., & Waldorp, L. J. (2018). How well do network models predict observations? On the importance of predictability in network models. *Behavior Research Methods*, 50(2), 853–861. [doi:10.3758/s134280170910x](https://doi.org/10.3758/s134280170910x)

Examples

```

set.seed(42)
mat <- matrix(rnorm(60), ncol = 4)
colnames(mat) <- LETTERS[1:4]
net <- build_network(as.data.frame(mat), method = "glasso")
predictability(net)

```

predict_links

*Predict Missing or Future Links in a Network***Description**

Computes link prediction scores for all node pairs using one or more structural similarity methods. Accepts netobject, mcml, cograph_network, or a raw weight matrix.

All methods are fully vectorized using matrix operations - no loops. Supports both weighted and binary adjacency, directed and undirected networks.

Usage

```
predict_links(
  x,
  methods = c("common_neighbors", "resource_allocation", "adamic_adar", "jaccard",
    "preferential_attachment", "katz"),
  weighted = TRUE,
  top_n = NULL,
  exclude_existing = TRUE,
  include_self = FALSE,
  katz_damping = NULL
)

## S3 method for class 'net_link_prediction'
print(x, ...)

## S3 method for class 'net_link_prediction'
summary(object, ...)
```

Arguments

x	A netobject, mcml, cograph_network, or numeric square matrix. For the print() method: an object of class net_link_prediction.
methods	Character vector. One or more of: "common_neighbors", "resource_allocation", "adamic_adar", "jaccard", "preferential_attachment", "katz". Default: all six methods.
weighted	Logical. If TRUE, use the weight matrix directly instead of binarizing. Default: TRUE.
top_n	Integer or NULL. Return only the top N predictions per method. Default: NULL (all pairs).
exclude_existing	Logical. If TRUE, exclude node pairs that already have an edge. Default: TRUE.
include_self	Logical. If TRUE, include self-loop predictions. Default: FALSE.
katz_damping	Numeric or NULL. Attenuation factor for Katz index. If NULL, auto-computed as $0.9 / \text{spectral_radius}(A)$. Default: NULL.

... In `print.net_link_prediction()` and `summary.net_link_prediction()`: Additional arguments (ignored).

object For the `summary()` method: an object of class `net_link_prediction`.

Details

Methods:

common_neighbors Number of shared neighbors. For directed graphs, sums shared out-neighbors and shared in-neighbors. Vectorized as $A \%*\% t(A) + t(A) \%*\% A$.

resource_allocation Zhou et al. (2009). Like common neighbors but weights each shared neighbor z by $1/\text{degree}(z)$. Penalizes hubs, rewards rare shared connections.

adamic_adar Adamic & Adar (2003). Like resource allocation but weights by $1/\log(\text{degree}(z))$. Less aggressive penalty than RA.

jaccard Ratio of shared neighbors to total neighbors. For directed graphs, computed on combined (out+in) neighbor sets.

preferential_attachment Product of source out-degree and target in-degree. Captures the "rich-get-richer" effect.

katz Katz (1953). Weighted sum of all paths between nodes, exponentially damped by path length. Computed via matrix inversion: $(I - \text{beta} * A)^{-1} - I$. Captures global structure.

Value

An object of class "net_link_prediction" containing:

predictions Data frame, one row per (node pair, method), with columns `from`, `to`, `method`, `score`, `existing` (was the pair already an edge?) and `rank`. Sorted by `score` (descending) within each method.

consensus Data frame, one row per node pair, with columns `from`, `to`, `avg_rank`, `n_methods` and `consensus_rank`, ordered by `avg_rank`. NULL when only one method was requested.

scores Named list of score matrices (one per method).

adjacency Integer 0/1 adjacency matrix of the input network.

methods Character vector of methods used.

nodes Character vector of node names.

directed Logical.

weighted Logical.

n_nodes Integer.

n_existing Integer. Number of existing edges.

In `print.net_link_prediction()`: The input object, invisibly.

In `summary.net_link_prediction()`: A data frame, one row per method, with columns `method`, `n_predictions`, `score_mean`, `score_sd`, `score_max` and `score_min`. A method with no predictions (every possible link already exists) has `n_predictions = 0` and NA scores.

References

- Liben-Nowell, D. & Kleinberg, J. (2007). The link-prediction problem for social networks. *JASIST*, 58(7), 1019–1031.
- Zhou, T., Lu, L. & Zhang, Y.-C. (2009). Network topology and link prediction. *European Physical Journal B*, 71, 623–630.
- Adamic, L. A. & Adar, E. (2003). Friends and neighbors on the Web. *Social Networks*, 25(3), 211–230.
- Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika*, 18(1), 39–43.
- Jaccard, P. (1901). Etude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Societe Vaudoise des Sciences Naturelles*, 37, 547–579.
- Barabasi, A.-L. & Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439), 509–512.

See Also

[evaluate_links](#) for prediction evaluation, [build_network](#) for network estimation.

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "C", "D", "A", "C", "E", "B"),
  V2 = c("B", "C", "D", "E", "C", "E", "A", "D"),
  V3 = c("C", "D", "E", "A", "D", "A", "B", "E")
)
net <- build_network(seqs, method = "relative")
pred <- predict_links(net)
print(pred)
summary(pred)
```

prepare

Prepare Event Log Data for Network Estimation

Description

Converts event log data (actor, action, time) into wide sequence format suitable for [build_network](#). Automatically parses timestamps, detects sessions from time gaps, and handles tie-breaking.

Usage

```
prepare(
  data,
  actor,
  action,
  time = NULL,
```

```

    order = NULL,
    session = NULL,
    time_threshold = 900,
    custom_format = NULL,
    is_unix_time = FALSE,
    unix_time_unit = c("seconds", "milliseconds", "microseconds"),
    timezone = "UTC"
)

## S3 method for class 'nestimate_data'
print(x, ...)

```

Arguments

<code>data</code>	Data frame with event log columns.
<code>actor</code>	Character or character vector. Column name(s) identifying who performed the action (e.g. "student" or <code>c("student", "group")</code>). If missing, all data is treated as one actor.
<code>action</code>	Character. Column name containing the action/state/code.
<code>time</code>	Character or NULL. Column name containing timestamps. Supports ISO8601, Unix timestamps (numeric), and 40+ date/time formats. If NULL, row order defines the sequence. Default: NULL.
<code>order</code>	Character or NULL. Column name for tie-breaking when timestamps are identical. If NULL, original row order is used. Default: NULL.
<code>session</code>	Character, character vector, or NULL. Column name(s) for explicit session grouping (e.g. "course" or <code>c("course", "semester")</code>). When combined with <code>time</code> , sessions are further split by time gaps. Default: NULL.
<code>time_threshold</code>	Numeric or FALSE. Maximum gap in seconds between consecutive events before a new session starts. Only used when <code>time</code> is provided. Set to FALSE to switch session-interval splitting off, so each actor (or actor-session) forms a single sequence however long the gaps are. Default: 900 (15 minutes).
<code>custom_format</code>	Character or NULL. Custom <code>strptime</code> format string for parsing timestamps. Default: NULL (auto-detect).
<code>is_unix_time</code>	Logical. If TRUE, treat numeric time values as Unix timestamps. Default: FALSE (auto-detected for numeric columns).
<code>unix_time_unit</code>	Character. Unit for Unix timestamps: "seconds", "milliseconds", or "microseconds". Default: "seconds".
<code>timezone</code>	Character. An Olson time zone (see OlsonNames) used to interpret timestamps that carry no zone information, and in which Unix timestamps are expressed. Timestamps that end in Z, UTC, GMT or a numeric offset such as +02:00 are converted from that offset. Parsing is therefore independent of the machine's local time zone. Default: "UTC".
<code>x</code>	For the <code>print()</code> method: an object of class <code>nestimate_data</code> .
<code>...</code>	In <code>print.nestimate_data()</code> : Additional arguments (ignored).

Details

Sessions are identified by the observed combinations of the actor and session columns, so identifiers containing separator characters stay distinct and high-cardinality identifiers cannot overflow. Missing values in any grouping column raise an error: drop or relabel those events first.

Value

A list with class "nestimate_data" containing:

sequence_data Data frame in wide format (one row per session, columns T1, T2, ...).

long_data The processed long-format data with session IDs.

meta_data Session-level metadata, one row per session in the row order of sequence_data: .session_id, .session_label, the actor column, the session column(s) under their own names, and every other column aggregated per session.

time_data Parsed time values in wide format (if time provided).

statistics List with total_sessions, total_actions and max_sequence_length, plus unique_actors only when actor was supplied (with no actor every row belongs to one synthetic actor, so the count would be meaningless).

In print.nestimate_data(): The input object, invisibly.

See Also

[build_network](#), [prepare_onehot](#)

Examples

```
set.seed(1)
df <- data.frame(
  student = rep(1:3, each = 5),
  code = sample(c("read", "write", "test"), 15, replace = TRUE),
  timestamp = seq.POSIXt(as.POSIXct("2024-01-01"), by = "min", length.out = 15)
)
prepared <- prepare(df, actor = "student", action = "code",
  time = "timestamp")
net <- build_network(prepared$sequence_data, method = "relative")
```

Description

Prepare simulated or real data for use with `tna::tna()` and related functions. Handles various input formats and ensures the output is compatible with TNA models.

Usage

```
prepare_for_tna(
  data,
  type = c("sequences", "long", "auto"),
  state_names = NULL,
  id_col = "Actor",
  time_col = "Time",
  action_col = "Action",
  validate = TRUE
)
```

Arguments

data	Data frame containing sequence data.
type	Character. Type of input data: "sequences" Wide format with one row per sequence (default). "long" Long format with one row per action. "auto" Automatically detect format based on column names.
state_names	Character vector. Expected state names, or NULL to extract from data. Default: NULL.
id_col	Character. Name of ID column for long format data. Default: "Actor".
time_col	Character. Name of time column for long format data. Default: "Time".
action_col	Character. Name of action column for long format data. Default: "Action".
validate	Logical. Whether to validate that all actions are in state_names. Default: TRUE.

Details

This function performs several preparations:

1. Converts long format to wide format if needed.
2. Validates that all actions/states are recognized.
3. Removes any non-sequence columns (e.g., id, metadata).
4. Converts factors to characters.
5. Ensures consistent column naming (V1, V2, ...).

Value

A data frame ready for use with TNA functions. For "sequences" type, returns a data frame where each row is a sequence and columns are time points (V1, V2, ...). For "long" type, converts to wide format first.

See Also

[wide_to_long](#), [long_to_wide](#) for format conversions.

Examples

```
# From wide format sequences
sequences <- data.frame(
  V1 = c("A", "B", "C", "A"), V2 = c("B", "C", "A", "B"),
  V3 = c("C", "A", "B", "C"), V4 = c("A", "B", "A", "B")
)
tna_data <- prepare_for_tna(sequences, type = "sequences")
```

```
prepare_onehot
```

```
Import One-Hot Encoded Data into Sequence Format
```

Description

Converts binary indicator (one-hot) data into the wide sequence format expected by [build_network](#) and `tna::tna()`. Each binary column represents a state; rows where the value is 1 are marked with the column name. Supports optional windowed aggregation.

Simultaneous active states are preserved using the same window-span representation as `tna::import_onehot()`: each input row/window is expanded to one sequence slot per code and transition counting occurs between windows, not between simultaneous states inside the same row.

Usage

```
prepare_onehot(
  data,
  cols,
  actor = NULL,
  session = NULL,
  interval = NULL,
  window_size = 3L,
  window_type = c("non-overlapping", "overlapping"),
  aggregate = FALSE
)
```

Arguments

<code>data</code>	Data frame with binary (0/1) indicator columns.
<code>cols</code>	Character vector. Names of the one-hot columns to use.
<code>actor</code>	Character or NULL. Name of the actor/ID column. If NULL, all rows are treated as a single sequence. Default: NULL.
<code>session</code>	Character or NULL. Name of the session column for sub-grouping within actors. Default: NULL.
<code>interval</code>	Integer or NULL. Number of rows per time point in the output. If NULL, all rows become a single time point group. Default: NULL.

window_size	Integer (≥ 1). Number of consecutive rows to aggregate into each window. Default: 3. Set window_size = 1 for no windowing (each row is its own time point).
window_type	Character. "non-overlapping" (fixed, separate windows) or "overlapping" (rolling, step = 1). Default: "non-overlapping".
aggregate	Logical. If TRUE, aggregate within each window by taking the first non-NA indicator per column. Default: FALSE.

Value

A data frame in wide format, one row per actor/session sequence, with columns named W<window>_T<slot> where each cell contains a state name or NA. Attributes windowed (always TRUE), window_size, window_span (the number of cols) and codes (the cols themselves) are set on the result.

See Also

[action_to_onehot](#) for the reverse conversion.

Examples

```
# Simple binary data
df <- data.frame(
  A = c(1, 0, 1, 0, 1),
  B = c(0, 1, 0, 1, 0),
  C = c(0, 0, 0, 0, 0)
)
seq_data <- prepare_onehot(df, cols = c("A", "B", "C"))

# With actor grouping
df$actor <- c(1, 1, 1, 2, 2)
seq_data <- prepare_onehot(df, cols = c("A", "B", "C"), actor = "actor")

# With windowing
seq_data <- prepare_onehot(df, cols = c("A", "B", "C"),
  window_size = 2, window_type = "non-overlapping")
```

q_analysis

*Q-Analysis***Description**

Computes Q-connectivity structure (Atkin 1974). Two maximal simplices are q-connected if they share a face of dimension $\geq q$. Reports:

- **Q-vector:** number of connected components at each q-level
- **Structure vector:** highest simplex dimension per node

Usage

```
q_analysis(sc)

## S3 method for class 'q_analysis'
print(x, ...)

## S3 method for class 'q_analysis'
plot(x, combined = TRUE, ...)
```

Arguments

sc	A simplicial_complex object.
x	For the print() and plot() methods: an object of class q_analysis.
...	In plot.q_analysis(): Ignored. In print.q_analysis(): Additional arguments (unused).
combined	When TRUE (default), the two panels are stitched side-by-side via gridExtra::arrangeGrob. When FALSE, returns a named list (q_vector, structure_vector) of ggplots.

Value

A q_analysis object with \$q_vector, \$structure_vector, and \$max_q.

In print.q_analysis(): The input object, invisibly.

In plot.q_analysis(): A grid grob (invisibly) when combined = TRUE; a named list of two ggplots when combined = FALSE.

Methods

- plot.q_analysis(): Two panels: Q-vector (components at each connectivity level) and structure vector (max simplex dimension per node).

References

Atkin, R. H. (1974). *Mathematical Structure in Human Affairs*.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
q_analysis(sc)
```

register_estimator	<i>Register a Network Estimator</i>
--------------------	-------------------------------------

Description

Register a custom or built-in network estimator function by name. Estimators registered here can be used by [build_network](#) via the method parameter.

Usage

```
register_estimator(name, fn, description, directed)
```

Arguments

name	Character. Unique name for the estimator (e.g. "relative", "glasso").
fn	Function. The estimator function. Must accept data as its first argument and ... for additional parameters. Must return a list with at least: matrix (square numeric matrix), nodes (character vector), directed (logical).
description	Character. Short description of the estimator.
directed	Logical. Whether the estimator produces directed networks.

Value

Invisible NULL.

See Also

[get_estimator](#), [list_estimators](#), [remove_estimator](#), [build_network](#)

Examples

```
my_fn <- function(data, ...) {  
  m <- cor(data)  
  diag(m) <- 0  
  list(matrix = m, nodes = colnames(m), directed = FALSE)  
}  
register_estimator("my_cor", my_fn, "Custom correlation", directed = FALSE)  
df <- data.frame(A = rnorm(20), B = rnorm(20), C = rnorm(20))  
net <- build_network(df, method = "my_cor")  
remove_estimator("my_cor")
```

remove_estimator	<i>Remove a Registered Estimator</i>
------------------	--------------------------------------

Description

Remove a network estimator from the registry.

Usage

```
remove_estimator(name)
```

Arguments

name	Character. Name of the estimator to remove.
------	---

Value

Invisible NULL.

See Also

[register_estimator](#), [list_estimators](#)

Examples

```
register_estimator("test_est", function(data, ...) diag(3),
  description = "test", directed = FALSE)
remove_estimator("test_est")
```

rename_models	<i>Rename the models of a netobject_group</i>
---------------	---

Description

Replaces the names of the constituent networks in a netobject_group (or any object inheriting from it). Useful when build_network() produced generic labels (e.g. "Cluster 1", "Cluster 2") and you want to substitute meaningful ones (e.g. "High engagement", "Low engagement").

Usage

```
rename_models(x, new_names)

## S3 method for class 'netobject_group'
rename_models(x, new_names)

## Default S3 method:
rename_models(x, new_names)
```

Arguments

x A `netobject_group` (or any object inheriting from it, such as `net_mlvar`).

new_names A character vector of new names. Must have the same length as `x`, contain no NA or empty strings, and be unique.

Value

A `netobject_group` of the same class and length as `x`, with `names()` replaced by `new_names`. The constituent networks are returned unchanged.

Examples

```
grp <- build_network(group_regulation_long, method = "tna",
                    actor = "Actor", action = "Action", time = "Time",
                    group = "Achiever")
names(grp)
names(rename_models(grp, c("High achievers", "Low achievers")))
```

sequence_compare	<i>Compare Subsequence Patterns Between Groups</i>
------------------	--

Description

Extracts all `k`-gram patterns (subsequences of length `k`) from sequences in each group, computes standardized residuals against the independence model, and optionally runs a permutation or chi-square test of group differences.

Usage

```
sequence_compare(
  x,
  group = NULL,
  sub = 3:5,
  min_freq = 5L,
  test = c("permutation", "chisq", "none"),
  iter = 1000L,
  adjust = "fdr"
)

## S3 method for class 'net_sequence_comparison'
print(x, ...)

## S3 method for class 'net_sequence_comparison'
summary(object, ...)

## S3 method for class 'net_sequence_comparison'
plot(
```

```

x,
top_n = 10L,
style = c("auto", "pyramid", "heatmap"),
sort = c("statistic", "frequency"),
alpha = 0.05,
show_residuals = FALSE,
...
)

```

Arguments

x	A netobject_group (from grouped build_network), a netobject (requires group), or a wide-format data.frame (requires group). For the print() and plot() methods: an object of class net_sequence_comparison.
group	Character or vector. Column name or vector of group labels. Not needed for netobject_group.
sub	Integer vector. Pattern lengths to analyze. Default: 3:5.
min_freq	Integer. Minimum frequency in each group for a pattern to be included: a pattern is kept only when its count reaches this threshold in <i>every</i> group. Default: 5.
test	Character. Inference method: one of "permutation" (default), "chisq", or "none". See Details.
iter	Integer. Permutation iterations. Only used when test = "permutation". Default: 1000.
adjust	Character. P-value correction method (see p.adjust). Default: "fdr".
...	In plot.net_sequence_comparison(), print.net_sequence_comparison() and summary.net_sequence_comparison(): Additional arguments (ignored).
object	For the summary() method: an object of class net_sequence_comparison.
top_n	Integer. Show top N patterns. Default: 10.
style	Character. "auto" (default) draws the back-to-back pyramid for exactly 2 groups and the heatmap for any other number; "pyramid" and "heatmap" force a specific style.
sort	Character. "statistic" (default) ranks patterns by test statistic or residual magnitude. "frequency" ranks by total occurrence count across all groups.
alpha	Numeric. Significance threshold for p-value display in the pyramid: patterns with $p_value < \alpha$ are starred and drawn in bold dark text, the rest stay plain grey. Default: 0.05.
show_residuals	Logical. If TRUE, print the standardized residual value inside each pyramid bar. Default: FALSE. Ignored for the heatmap (which always shows residuals).

Details

Standardized residuals are always computed from a 2xG contingency table of (this pattern vs. everything else) using the textbook formula $(o - e) / \sqrt{e * (1 - r/N) * (1 - c/N)}$. They describe how much each group's count for a given pattern deviates from expectation under independence, scaled to be approximately $N(0,1)$ under the null.

The optional test argument chooses an inference method:

"permutation" Shuffles group labels across sequences and recomputes a per-pattern statistic (row-wise Euclidean residual norm). Answers: "is this pattern's distribution associated with group membership at the *actor* level?" Respects the sequence as the unit of analysis; can be under-powered when the number of sequences is small.

"chisq" Runs `chisq.test` on the 2xG table per pattern. Answers: "do the group *streams* generate this pattern at different rates?" Treats each k-gram occurrence as an event; fast and powerful even with few sequences, but the iid assumption it makes is optimistic when sequences are strongly autocorrelated.

"none" Skip inference. Only residuals, frequencies, and proportions are returned.

P-values are adjusted once across all patterns (not per-pattern) using any method supported by `p.adjust`. The default is "fdr" (Benjamini-Hochberg).

Value

An object of class "net_sequence_comparison" containing:

patterns Tidy data.frame, one row per retained k-gram pattern. Always present: pattern, length, and one `freq_<group>`, `prop_<group>` and `resid_<group>` column per group. If `test = "permutation"`: `effect_size`, `p_value`. If `test = "chisq"`: `statistic`, `p_value`. Rows are ordered by ascending adjusted `p_value` when a test was run, and by descending maximum absolute residual otherwise.

groups Character vector of group names, sorted.

n_patterns Integer. Number of rows in patterns, i.e. the patterns meeting `min_freq` in every group.

params List of sub, `min_freq`, `test`, `iter`, `adjust`.

In `print.net_sequence_comparison()`: The input object, invisibly.

In `summary.net_sequence_comparison()`: The patterns data.frame: tidy, one row per k-gram pattern, with a frequency, proportion and standardized-residual column per group, and the test columns when test was not "none".

In `plot.net_sequence_comparison()`: The drawn ggplot object, invisibly (the plot is also printed). NULL, invisibly, when the object holds no patterns.

Plot styles

Visualizes pattern-level standardized residuals across groups. Two styles are available, and `style = "auto"` (the default) picks between them by the number of groups:

"pyramid" Back-to-back bars of pattern proportions, shaded by each side's standardized residual. Requires exactly 2 groups; an explicit `style = "pyramid"` on any other number is an error.

"heatmap" One tile per (pattern, group) cell, colored by standardized residual. Works for any number of groups.

Residuals are read directly from the `resid_<group>` columns in `$patterns`, which are always populated regardless of the inference method chosen in `sequence_compare`.

References

- Haberman, S. J. (1973). The analysis of residuals in cross-classified tables. *Biometrics*, 29(1), 205–220. (standardized residuals)
- Benjamini, Y. & Hochberg, Y. (1995). Controlling the false discovery rate. *Journal of the Royal Statistical Society B*, 57(1), 289–300. (the default adjust = "fdr")

Examples

```
set.seed(1)
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 60, TRUE),
  V2 = sample(LETTERS[1:4], 60, TRUE),
  V3 = sample(LETTERS[1:4], 60, TRUE),
  V4 = sample(LETTERS[1:4], 60, TRUE)
)
grp <- rep(c("X", "Y"), 30)
net <- build_network(seqs, method = "relative")
res <- sequence_compare(net, group = grp, sub = 2:3, test = "chisq")
```

sequence_plot

Sequence Plot (heatmap, index, or distribution)

Description

Single entry point for three categorical-sequence visualisations.

- type = "heatmap" (default): dense carpet, rows reordered by sort / dendrogram (single panel).
- type = "index": same data layout, but rows separated by thin gaps (no dendrogram). Supports grouping via group or a net_clustering, plus a ncol x nrow facet grid.
- type = "distribution": dispatches to [distribution_plot](#).

Usage

```
sequence_plot(
  x,
  type = c("heatmap", "index", "distribution"),
  sort = c("lcs", "frequency", "start", "end", "hamming", "osa", "lv", "dl", "qgram",
    "cosine", "jaccard", "jw"),
  tree = NULL,
  group = NULL,
  scale = c("proportion", "count"),
  geom = c("area", "bar"),
  na = TRUE,
  normalize = FALSE,
  trim = NULL,
```

```

panel = c("both", "summary", "channels"),
expand = NULL,
combine = NULL,
rest = c("clusters", "pooled", "none"),
rest_label = "Other states",
trim_clusterwise = FALSE,
row_gap = 0,
dendrogram_width = 1.2,
k = NULL,
k_color = "white",
k_line_width = 2.5,
state_colors = NULL,
na_color = "grey90",
cell_border = NA,
frame = FALSE,
width = NULL,
height = NULL,
main = NULL,
show_n = TRUE,
time_label = "Time",
xlab = NULL,
y_label = NULL,
ylab = NULL,
tick = NULL,
ncol = NULL,
nrow = NULL,
combined = TRUE,
legend = NULL,
legend_size = NULL,
legend_title = NULL,
legend_ncol = NULL,
legend_border = NA,
legend_bty = "n"
)

## S3 method for class 'mcm1_sequence_plot'
print(x, ...)

```

Arguments

x Wide-format sequence data. Accepts:

- data.frame / matrix** Rows = sequences, columns = time points.
- netobject** Extracts \$data.
- net_clustering** From [build_clusters](#). Uses \$data, \$assignments for grouping, and \$distance for dendrogram.
- netobject_group** From [cluster_network](#) or [build_network](#) on a clustering. Extracts data and assignments from attr(, "clustering").

net_mmm From `build_mmm`. Uses `$data` (falling back to `$models[[1]]$data`) and `$assignments`.

tna From the `tna` package. Decodes integer-encoded sequences.

mcml From `build_mcml` (built from sequences). Produces a **multichannel** plot: one panel per cluster plus a macro Summary panel. `type = "heatmap"/"index"` draw the carpet (each channel's own states solid, other clusters a faded wash); `type = "distribution"` draws the stacked distribution (add `normalize = TRUE` for a TraMineR-style `seqdplot` where each time point sums to 1). See the section *Multichannel view of an mcml* for the options that shape it, and *Value* for what it returns.

For the `print()` method: an object of class `mcml_sequence_plot`.

<code>type</code>	One of "heatmap" (default), "index", or "distribution".
<code>sort</code>	Row-ordering strategy for heatmap / within-panel for index. One of "lcs" (default), "frequency", "start", "end", or any <code>build_clusters</code> distance ("hamming", "osa", "lv", "dl", "qgram", "cosine", "jaccard", "jw").
<code>tree</code>	Optional <code>hclust</code> /dendrogram/ <code>agnes</code> object to supply row ordering (heatmap only; overrides <code>sort</code>).
<code>group</code>	Optional grouping vector (length <code>nrow(x)</code>) producing one facet per group. Index/distribution only. Ignored for heatmap.
<code>scale, geom, na</code>	Passed to <code>distribution_plot</code> when <code>type = "distribution"</code> . For an <code>mcml</code> (<code>type = "distribution"</code>), <code>na = FALSE</code> drops the NA (ended) band and shows every time point as shares of the sequences still running there, so each panel stacks to 100 percent (cluster panels only with <code>rest = "clusters"</code> or "pooled"; a time point where no sequence is running stays empty).
<code>normalize</code>	<code>mcml + type = "distribution"</code> only. When <code>TRUE</code> , each time point is normalised to sum to 1 within its channel (TraMineR-style <code>seqdplot</code> composition); when <code>FALSE</code> (default) the stack shows prevalence and is capped with an NA band.
<code>trim</code>	Optional time-axis truncation, to stop a few long sequences from stretching the plot. Applies to all three types (including the <code>mcml</code> multichannel view). <code>NULL</code> (default) plots the full width. A fraction in $(0, 1)$ drops everything past that quantile of sequence lengths (e.g. <code>trim = 0.95</code> keeps the columns covering the shortest 95% of sequences); a value ≥ 1 is an absolute cut (<code>trim = 50</code> keeps the first 50 time points).
<code>panel</code>	<code>mcml + type = "distribution"</code> only. Which panel to draw. "both" (default) stacks the macro Summary channel and the per-cluster channels on one figure, each with its own legend; "summary" draws the macro channel alone, keyed and coloured by cluster; "channels" draws the per-cluster channels alone, keyed and coloured by state. The macro channel is keyed by cluster and the rest by state, so a cluster and a state can land on the same colour – drawing one panel avoids that and gives it a default title.
<code>expand</code>	For an <code>mcml</code> , names of clusters whose member states are shown individually in the Summary band; "all" or <code>TRUE</code> expands every cluster. The per-cluster channels are unaffected. Default <code>NULL</code> keys the Summary band by cluster.
<code>combine</code>	For an <code>mcml</code> , clusters to merge into one channel. A character vector merges one group (e.g. <code>combine = c("Cognitive", "Affective")</code>); a list merges several,

and its names label the merged channels (default label "Cognitive + Affective"). A merged group acts as one cluster throughout the figure: one per-cluster panel holding all its states, one key in the Summary band, and one faded band in the other panels. `expand` is resolved after merging, so it can name the merged label. Errors on unknown clusters, a group of fewer than two, or a cluster in two groups. Default NULL draws the partition as built.

<code>rest</code>	For an <code>mcml</code> , how a cluster's panel shows the time its subjects spend in <i>other</i> clusters. "clusters" (default): one faded band (or wash, in the carpet) per other cluster. "pooled": all other clusters as one grey band labelled <code>rest_label</code> . "none": left blank, so the panel shows only its own states; in the distribution view the NA (ended) band is dropped too, and the panel's height at each time point is the share of subjects in that cluster. Ignored with <code>normalize = TRUE</code> , which rescales each panel to its own states. The Summary panel is unaffected.
<code>rest_label</code>	For an <code>mcml</code> , the legend text for time spent in other clusters. Default "Other states"; e.g. "Others" or "Rest of states". The pooled band (<code>rest = "pooled"</code>) takes it as is; the per-cluster bands read "Social (Other states)". Must not equal a state or cluster name.
<code>trim_clusterwise</code>	Grouped type = "index" / "distribution" only, and only when <code>trim</code> is a fraction. FALSE (default) computes one cutoff on the pooled data and applies it to every panel, so all facets share the same width and the time axes stay aligned. TRUE crops each group to its own length quantile, so panels can end up at different widths (ragged axes). Absolute trim (≥ 1) ignores this - the column is the same everywhere either way.
<code>row_gap</code>	Fraction of row height used as vertical gap between sequences in index plots. 0 (default) = dense like heatmap. Try 0.15 for visible separators at low row counts.
<code>dendrogram_width</code>	Width ratio of the dendrogram panel (heatmap).
<code>k</code>	Optional integer. When supplied in <code>type = "heatmap"</code> , cuts the dendrogram into <code>k</code> clusters and draws thin horizontal separators between them in the carpet. Ignored when there is no dendrogram (e.g. <code>sort = "start"</code>) or for other types.
<code>k_color</code>	Colour for the cluster separator lines. Default "white".
<code>k_line_width</code>	Line width for the cluster separators. Default 2.5.
<code>state_colors</code>	Colours for the fill keys. Two forms: <i>unnamed</i> - one colour per state, in level order (states are ordered as <code>sort(unique(...))</code>); <i>named</i> - a lookup, where only the keys you name are overridden and every other key keeps its default. Names this figure does not draw are dropped with a message naming them, so one project-wide palette can be handed to every plot and each takes the keys that apply to it. For an <code>mcml</code> the named form reaches the whole figure, not just the states: a cluster name colours its Summary band, its channel strip and its faded band in the other panels, and a group merged by <code>combine</code> is named by its label (the list name you gave it, or "A + B"). <code>rest_label</code> is a key too. So <code>state_colors = c(plan = "#0072B2", "Planning + Monitoring" = "#D55E00")</code> recolours one state and one combined cluster and leaves the rest of the palette alone.

na_color	Colour for NA cells.
cell_border	Cell border colour. NA (default) = off.
frame	FALSE (default) draws no box - axis ticks and labels still appear. TRUE draws a box around each panel.
width, height	Optional device dimensions in inches. When supplied, opens a new graphics device via <code>grDevices::dev.new()</code> . In knitr chunks use the <code>fig.width / fig.height</code> chunk options instead.
main	Plot title.
show_n	Append "(n = N)" to the title.
time_label, xlab	X-axis label. xlab is an alias.
y_label, ylab	Y-axis label (distribution only). ylab alias.
tick	Show every Nth x-axis label. NULL = auto.
ncol, nrow	Facet grid dimensions (index + distribution). Ignored when combined = FALSE.
combined	Index and distribution types only. When TRUE (default), groups are arranged on one figure via <code>graphics::layout()</code> . When FALSE, each group is drawn on its own page (one full-size figure per group, with its own legend). Single-group calls (<code>G == 1</code>) ignore this argument. Heatmap is always single-figure.
legend	Legend position: "bottom", "right", or "none". NULL (default) resolves to "right" for every type.
legend_size	Legend text size. NULL (default) auto-scales from the device width so the legend looks proportional at 5 in vs 12 in figures (clamped to <code>[0.65, 1.2]</code>).
legend_title	Optional legend title.
legend_ncol	Number of legend columns.
legend_border	Swatch border colour.
legend_bty	"n" or "o".
...	In <code>print.mcml_sequence_plot()</code> : Ignored.

Value

An `mcml` input returns the multichannel figure: one panel per channel (the macro Summary and one per cluster), stacked, each with its own legend of its own clusters or states. When more than one channel is drawn this is an `mcml_sequence_plot` (a `gtable` whose `print` method draws it); when `panel = "summary"` leaves a single channel it is a plain `ggplot`. Every other input draws with base graphics and returns, invisibly, a list whose shape depends on type:

"heatmap" `ord` (integer row order actually plotted), `codes` (the integer-encoded, trimmed sequence matrix), `palette`, `levels` (state labels, parallel to `palette`), and `sort_used` (the ordering strategy applied, "net_clustering" when a clustering dendrogram was used).

"index" `codes`, `palette`, `levels`, `orders` (list of integer row orders, one per panel, indexing the original rows) and `groups` (panel labels).

"distribution" Whatever `distribution_plot` returns: counts, proportions, levels, palette, groups.

In `print.mcml_sequence_plot()`: `x`, invisibly. Called for the side effect of drawing it on a new page of the current graphics device.

Multichannel view of an mcml

An mcml built from sequences stores, for every cluster, the full sequence matrix with the other clusters' states blanked out. Each cluster is therefore a *channel*, and `sequence_plot()` stacks them:

Summary The macro sequence: at every time point, the cluster each subject is in.

One panel per cluster That cluster's own states, plus the time its subjects spend in the other clusters.

The options apply in this order, so each one sees the result of the one before:

1. `combine` merges clusters into one channel. The merged group is then one cluster throughout the figure: one panel with all its states, one Summary key, one band in the other panels. The object itself is not changed.
2. `expand` opens clusters (including a merged group, by its label) into their member states in the Summary panel only.
3. `rest` and `rest_label` set how a cluster's panel shows the other clusters: one faded band per cluster labelled "<cluster> (<rest_label>)" (`rest = "clusters"`), one grey band labelled `rest_label` ("pooled"), or nothing ("none").
4. `na` (distribution only) keeps the NA band of sequences that have ended (TRUE, shares of all subjects) or drops it (FALSE, shares of the subjects still running, so every panel stacks to 100 percent unless `rest = "none"`). `normalize = TRUE` instead rescales each cluster panel to its own states, which ignores `rest` and `na`.

`panel` draws the Summary or the cluster panels alone, and `trim` cuts the time axis for every panel at once.

Methods

- `print.mcml_sequence_plot()`: Print method for the figure `sequence_plot` returns for an mcml with more than one channel: one panel per channel (the macro Summary and one per cluster), each with its own legend.

See Also

[distribution_plot](#), [build_clusters](#), [build_mcml](#)

Examples

```
sequence_plot(trjectories)
```

```
sequence_plot(trjectories, type = "index")
sequence_plot(trjectories, type = "distribution")
```

```
# Multichannel MCML view: one channel per cluster + a macro Summary.
fit <- build_mcml(
  group_regulation_long,
  clusters = list(Cognitive = c("discuss", "synthesis", "consensus", "cohesion"),
                  Regulation = c("plan", "monitor", "adapt", "coregulate"),
```

```

        Affective = "emotion"),
    actor = "Actor", action = "Action", time = "Time")
sequence_plot(fit) # multichannel carpet

# Shape the multichannel view (see the section above).
sequence_plot(fit, type = "distribution",
              combine = list(Task = c("Cognitive", "Regulation")),
              expand = "Task") # merge, then open

# Colour by name: one state, one cluster, one combined group. Everything
# not named keeps its default colour.
sequence_plot(fit, type = "distribution",
              combine = list(Task = c("Cognitive", "Regulation")),
              state_colors = c(Task = "#0072B2", Affective = "#D55E00",
                              emotion = "#CC79A7"))

```

session_ids

The session behind each sequence

Description

Names every sequence of a network or of a clustering by the columns it was built from. A network built from long data with actor and session has one sequence per actor-session; its rows are ordered by the grouping, not by the input, and a fitted clustering reports its assignments in that same order. `session_ids()` returns the key that joins them back to the input data, so no label has to be parsed.

Usage

```

session_ids(x, ...)

## Default S3 method:
session_ids(x, ...)

## S3 method for class 'netobject'
session_ids(x, ...)

## S3 method for class 'net_mmm'
session_ids(x, ...)

## S3 method for class 'net_clustering'
session_ids(x, ...)

```

Arguments

x	A netobject from build_network on long data, or a net_mmm (build_mmm) or net_clustering (build_clusters) fitted on such a network.
...	Unused.

Value

A data frame with one row per sequence, in the row order of the network's \$data (and of the fit's assignments):

sequence Integer row number of the sequence.

actor and session columns The actor and session columns given to build_network(), under their own names and with their own values.

session_label The readable label of the sequence. With time, sessions split at time gaps carry a "s<n>" suffix, so this column separates them.

cluster For a net_mmm or net_clustering: the assigned cluster (integer).

posterior For a net_mmm: the posterior probability of the assigned cluster.

Errors

Raises nestimate_no_session_ids when x carries no per-sequence metadata: a network built from wide data, a fit on wide data or on a tna model, or a fit made before Nestimate 0.9.6 (refit it). Raises nestimate_session_ids_misaligned when the metadata and the sequences differ in number.

See Also

[build_network](#), [build_mmm](#), [build_clusters](#)

Examples

```
events <- data.frame(
  student = rep(c("s1", "s2", "s3"), each = 8),
  step    = rep(c("a", "b"), each = 4, times = 3),
  action  = sample(c("read", "write", "test"), 24, replace = TRUE)
)
net <- build_network(events, actor = "student", session = "step",
  action = "action", method = "relative")
session_ids(net)

fit <- build_mmm(net, k = 2, n_starts = 2, seed = 1)
session_ids(fit)
```

set_state_colors

Set the state colours carried by a network object

Description

Attaches a palette to the object so every figure drawn from it uses the same colours: [sequence_plot](#), [distribution_plot](#), [plot_state_frequencies](#) and `cograph::splot()`.

Usage

```

set_state_colors(x, colors)

## Default S3 method:
set_state_colors(x, colors)

## S3 method for class 'netobject'
set_state_colors(x, colors)

## S3 method for class 'htna'
set_state_colors(x, colors)

## S3 method for class 'mcml'
set_state_colors(x, colors)

## S3 method for class 'netobject_group'
set_state_colors(x, colors)

state_colors(x) <- value

```

Arguments

<code>x</code>	A <code>netobject</code> , <code>netobject_group</code> , <code>mcml</code> or <code>htna</code> .
<code>colors</code>	A named character vector of colours, e.g. <code>c(plan = "#0072B2", monitor = "#D55E00")</code> . Names are states, and for an <code>mcml</code> may also be cluster names. Names the object does not carry are dropped with a message, so one project-wide palette can be attached to every object. States you do not name keep the default Okabe-Ito colour. <code>NULL</code> removes a palette set earlier.
<code>value</code>	The palette, as for <code>colors</code> , in the replacement form <code>state_colors(x) <- value</code> .

Value

`x`, with the palette stored in `x$state_colors` and, for an object carrying `$nodes`, mirrored into `x$meta$plot$defaults$node_fill` in node order so `cograph::splot()` honours it. The class is unchanged.

See Also

[state_colors](#) to read the resolved palette back.

Examples

```

net <- build_network(group_regulation_long, method = "relative",
                    actor = "Actor", action = "Action", time = "Time")
net <- set_state_colors(net, c(plan = "#0072B2", monitor = "#D55E00"))
state_colors(net)

```

simplicial_degree	<i>Simplicial Degree</i>
-------------------	--------------------------

Description

Counts how many simplices of each dimension contain each node.

Usage

```
simplicial_degree(sc, normalized = FALSE)
```

Arguments

sc	A simplicial_complex object.
normalized	Divide by maximum possible count. Default FALSE.

Value

Data frame with node, columns d0 through d_k, and total (sum of d1+). Sorted by total descending.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
simplicial_degree(sc)
```

simplicial_features	<i>Tidy Topological Features for One or Many Networks</i>
---------------------	---

Description

Builds a simplicial complex per network and returns its topological summaries as a tidy data.frame – one row per network, one column per feature – ready to use as regression predictors or to join onto unit-level outcomes.

Usage

```
simplicial_features(
  x,
  threshold = 0,
  max_dim = 4L,
  normalize = FALSE,
  type = "clique"
)
```

Arguments

x	A netobject, netobject_group, mcm1, a square weight matrix, or a named list of any of these. A group or list yields one row per member, an mcm1 one row per cluster, and a single network one row.
threshold	Minimum absolute edge weight for an edge to exist (passed to build_simplicial). Topology is a step function of this value, so a single threshold is a choice, not a result – pass a vector to sweep it and get one row per network per threshold.
max_dim	Maximum simplex dimension retained. Default 4.
normalize	Divide simplex counts by the number of nodes, so networks of different size are comparable. Default FALSE.
type	Complex type passed to build_simplicial . Default "clique".

Details

Higher-order structure is reported as d2, d3, ... : the number of simplices of that dimension. A 2-simplex is a triangle of three mutually connected states, a 3-simplex a tetrahedron of four. These count *co-participation in a dense region*, not statistical interaction.

Value

A data.frame with one row per network (per threshold), and columns network, threshold, n_nodes, n_edges, b0, b1 (Betti numbers), euler, max_q, d1 ... d<max_dim> (simplex counts by dimension), and higher_order (the total of d2 upward).

See Also

[build_simplicial](#), [betti_numbers](#), [q_analysis](#), [outcome_model](#)

Examples

```
m1 <- matrix(c(0, .6, .5, .6, 0, .4, .5, .4, 0), 3, 3,
             dimnames = list(c("A", "B", "C"), c("A", "B", "C")))
m2 <- matrix(c(0, .2, 0, .2, 0, .1, 0, .1, 0), 3, 3,
             dimnames = list(c("A", "B", "C"), c("A", "B", "C")))
simplicial_features(list(dense = m1, sparse = m2), threshold = 0.3)

# Sweep the threshold rather than committing to one.
simplicial_features(list(dense = m1), threshold = c(0.1, 0.3, 0.5))
```

Description

Simulated frequency counts of 9 self-regulated learning (SRL) strategies for 250 university students. Strategies are grouped into three clusters: metacognitive (Planning, Monitoring, Evaluating), cognitive (Elaboration, Organization, Rehearsal), and resource management (Help_Seeking, Time_Mgmt, Effort_Reg). Within-cluster correlations are moderate (0.3–0.6), cross-cluster correlations are weaker.

Usage

```
srl_strategies
```

Format

A data frame with 250 rows and 9 columns, one row per student. Every column is numeric (double) and holds a whole-number count of how often that student used the strategy; observed values range from 0 to 37.

Examples

```
net <- build_network(srl_strategies, method = "glasso",
                    params = list(gamma = 0.5))
net
```

state_colors

The state colours an object will draw with

Description

Reads back the palette an object resolves to: the colours set with [set_state_colors](#) plus the defaults filled in for everything else, so the table is what the figures actually use.

Usage

```
state_colors(x, ...)

## Default S3 method:
state_colors(x, ...)

## S3 method for class 'netobject'
state_colors(x, ...)

## S3 method for class 'htna'
state_colors(x, ...)

## S3 method for class 'mcm1'
state_colors(x, ...)
```

```
## S3 method for class 'netobject_group'
state_colors(x, ...)
```

Arguments

x A netobject, netobject_group, mcml or htna.
... Ignored, for method consistency.

Value

A data.frame, one row per colour key the object carries, with columns state (the key), color (the hex colour it draws with) and source ("set" when the palette named it, "default" when it fell back to Okabe-Ito). For an mcml the cluster names appear after the states.

See Also

[set_state_colors](#).

Examples

```
net <- build_network(group_regulation_long, method = "relative",
                    actor = "Actor", action = "Action", time = "Time")
state_colors(set_state_colors(net, c(plan = "#0072B2")))
```

state_distribution	<i>Per-Class State Distribution as a Tidy Data Frame</i>
--------------------	--

Description

Returns a tidy data.frame(group, state, count, proportion) with one row per (group, state) cell. Companion to [state_frequencies](#) (which counts unique states in raw sequence input); state_distribution() pulls the same shape of frame from a fitted Nestimate object so analyses don't have to reach for the underlying \$data slot directly.

Usage

```
state_distribution(x, ...)

## S3 method for class 'netobject'
state_distribution(x, ...)

## S3 method for class 'htna'
state_distribution(x, ...)

## S3 method for class 'mcml'
state_distribution(x, include_macro = FALSE, ...)
```

```
## S3 method for class 'netobject_group'
state_distribution(x, ...)
```

```
## Default S3 method:
state_distribution(x, ...)
```

Arguments

x	A netobject, netobject_group, mcml, or htna object.
...	Currently unused.
include_macro	For mcml: when TRUE, prepend a group = "macro" block aggregating across clusters. Ignored for the other classes.

Details

Used internally by [plot_state_frequencies](#) as the data layer behind every chart, and surfaced as the \$table slot of the returned state_freq object.

Value

A data.frame with one row per (group, state) cell and columns group (character), state (character), count (integer), and proportion (numeric, within-group share). A single ungrouped network yields a single group labelled "all".

Examples

```
data(group_regulation_long, package = "Nestimate")
net <- build_network(group_regulation_long, method = "frequency",
                    format = "long", actor = "Actor", action = "Action",
                    order = "Time", group = "Course")
state_distribution(net)
```

state_frequencies	<i>Compute State Frequencies from Trajectory Data</i>
-------------------	---

Description

Counts how often each state appears across all trajectories. Returns a data frame sorted by frequency (descending).

Usage

```
state_frequencies(data)
```

Arguments

data	A list of character vectors (trajectories) or a data.frame.
------	---

Value

A data frame with one row per distinct state, sorted by count (descending), with columns state, count and proportion (share of all observations, rounded to 4 decimal places).

Examples

```
trajs <- list(c("A","B","C"), c("A","B","A"))
state_frequencies(trajs)
```

subtract_networks	<i>Subtract one network from another</i>
-------------------	--

Description

Returns $x - y$ as a `netdifference` object: the element-wise difference of the two weight matrices. Works on any pair of networks; for an edge-betweenness difference, subtract two [net_edge_betweenness](#) results. Draw the signed difference network with `cograph::splot(d)` or `cograph::plot_difference(d)`; `cograph` handles the colouring and node palette.

Usage

```
subtract_networks(x, y)

## S3 method for class 'netdifference'
print(x, max_print = 12L, ...)
```

Arguments

<code>x, y</code>	A <code>netobject</code> , <code>cograph_network</code> , or numeric square matrix. Both must share the same nodes in the same order. For the <code>print()</code> method, <code>x</code> is the <code>netdifference</code> object.
<code>max_print</code>	Integer. Rows to show in <code>print()</code> . Default 12.
<code>...</code>	In <code>print.netdifference()</code> : Ignored.

Value

A `netdifference` object: a `netobject` whose `$weights` and `$difference_matrix` are $x - y$, carrying the source matrices `$x` and `$y`.

Examples

```
early <- data.frame(
  V1 = c("A","B","A","C"), V2 = c("B","C","B","A"),
  V3 = c("C","A","C","B"))
late <- data.frame(
  V1 = c("B","A","C","B"), V2 = c("C","B","A","C"),
```

```
V3 = c("A","C","B","A"))
a <- build_network(early, method = "relative")
b <- build_network(late, method = "relative")
subtract_networks(a, b)
# edge-betweenness difference:
subtract_networks(net_edge_betweenness(a), net_edge_betweenness(b))
```

trajectories	<i>Student Engagement Trajectories</i>
--------------	--

Description

Wide-format state sequences of student engagement over 15 weekly observations. Each row is one student; columns 1..15 hold the engagement state for that week. States: "Active", "Average", "Disengaged". Missing weeks are NA.

Usage

```
trajectories
```

Format

A character matrix with 138 rows and 15 columns, one row per student. Columns are named "1".."15" (the week). Entries are one of "Active", "Average", "Disengaged", or NA; NA runs at the end of a row mark drop-out, which the right-censored sequence verbs ([actor_endpoints](#), [mark_terminal_state](#)) are built to read.

Examples

```
sequence_plot(trajectories, main = "Engagement trajectories")
sequence_plot(trajectories, k = 3)
sequence_plot(trajectories, type = "distribution")
```

transition_entropy	<i>Transition Entropy of a Markov Chain</i>
--------------------	---

Description

Computes per-state branching entropy, stationary entropy, and the chain-level entropy rate of a Markov transition process. The entropy rate $H = - \sum_i \pi_i \sum_j P_{ij} \log_b P_{ij}$ (with π the stationary distribution from the eigendecomposition of $P^\top$ at $\lambda = 1$) is the Shannon-McMillan-Breiman per-step uncertainty of trajectories - the canonical information-theoretic summary of a transition matrix, introduced to behavioral research as gaze transition entropy by Krejtz et al. (2015) and tracked in real time as mobile transition matrix entropy by Krejtz et al. (2025). The normalized fields (*_norm, division by $\log_b n$) are the scale-free variants those papers report.

Usage

```

transition_entropy(x, base = 2, normalize = TRUE)

## S3 method for class 'net_transition_entropy'
print(x, digits = 3, ...)

## S3 method for class 'net_transition_entropy_group'
print(x, ...)

## S3 method for class 'net_transition_entropy'
summary(object, ...)

## S3 method for class 'summary.net_transition_entropy'
print(x, digits = 3, ...)

## S3 method for class 'net_transition_entropy'
plot(x, title = "Transition Entropy", fill = "#0072B2", ...)

```

Arguments

x	A netobject, cograph_network, tna object, row-stochastic numeric transition matrix, or a wide sequence data.frame (rows = actors, columns = time-steps; a relative transition network is built automatically). Group dispatch on netobject_group. For the print() and plot() methods: an object of class net_transition_entropy or net_transition_entropy_group (or its summary()).
base	Numeric. Logarithm base. 2 (default) for bits, exp(1) for nats, 10 for hartleys.
normalize	Logical. If TRUE (default), rows that do not sum to 1 are normalised automatically (with a warning).
digits	Integer. Digits to round numeric output. Default 3.
...	In plot.net_transition_entropy(), print.net_transition_entropy(), print.summary.net_tra and summary.net_transition_entropy(): Ignored. In print.net_transition_entropy_group(): Forwarded to print.net_transition_entropy.
object	For the summary() method: an object of class net_transition_entropy.
title	Character. Plot title.
fill	Character. Bar fill colour. Default Okabe-Ito blue.

Details

Convention $0 \log 0 := 0$ is applied, so absorbing or deterministic rows contribute zero per-row entropy. The chain need not be irreducible; π is computed from the eigendecomposition of $P^\top$ as elsewhere in the package. For non-ergodic chains the returned π is one stationary distribution among many - interpret with the help of [chain_structure](#).

The relation $h(P) \leq H(\pi)$ holds with equality iff successive states are independent. The deficit $H(\pi) - h(P)$ is reported as redundancy - a measure of how much memory the chain has at order 1.

Value

An object of class "net_transition_entropy" with:

row_entropy Named numeric vector, length n . Per-state branching entropy $H(P_{i.}) = -\sum_j P_{ij} \log P_{ij}$.

row_entropy_norm Named numeric vector. row_entropy divided by the ceiling $\log_b n$ (in $[0, 1]$; all zeros when $n = 1$).

stationary Named numeric vector. Stationary distribution π .

stationary_entropy Scalar. $H(\pi) = -\sum_i \pi_i \log \pi_i$ - the entropy of π treated as an i.i.d. distribution. Upper bound on the entropy rate.

stationary_entropy_norm Scalar. stationary_entropy divided by the ceiling $\log_b n$.

entropy_rate Scalar. $h(P) = \sum_i \pi_i H(P_{i.})$ - the Shannon-McMillan-Breiman entropy rate.

entropy_rate_norm Scalar. entropy_rate divided by the ceiling $\log_b n$.

redundancy Scalar. $H(\pi) - h(P)$, the entropy deficit attributable to serial dependence; zero for an i.i.d. chain (rows of P all equal π).

redundancy_norm Scalar. The *relative* redundancy $(H(\pi) - h(P))/H(\pi)$ (the fraction of the stationary entropy removed by order-1 memory), **not** redundancy divided by $\log_b n$; 0 when $H(\pi) = 0$.

max_entropy Scalar. The normalising ceiling $\log_b n$.

base Logarithm base used.

states Character vector of state names.

For a netobject_group the result is a "net_transition_entropy_group": a named list holding one such object per group.

In print.net_transition_entropy(), print.net_transition_entropy_group() and print.summary.net_transition_entropy_group(): Prints the object invisibly.

In summary.net_transition_entropy(): A summary.net_transition_entropy containing

table tidy per-state data.frame, sorted by contribution_pct descending

chain tidy chain-level data.frame with raw and normalised $h(P)$, $H(\pi)$, redundancy, and ceiling

base logarithm base used

In plot.net_transition_entropy(): A ggplot object.

Methods

- plot.net_transition_entropy(): Bar chart of per-state row entropy with overlaid horizontal lines at the entropy rate $h(P)$ (chain-level summary) and the maximum row entropy $\log_b n$ (uniform branching). Bar widths are proportional to the stationary probability so the visual area sums to the entropy rate.
- summary.net_transition_entropy(): Returns a tidy per-state contribution table sorted by share of the chain-level entropy rate (largest first), so the dominant contributors to $h(P)$ are visible at a glance. Each row contains the stationary mass, the raw and normalised row entropy, the additive contribution $\pi_i H(P_{i.})$, and that contribution as a percentage of $h(P)$.

References

- Cover, T.M. & Thomas, J.A. (2006). *Elements of Information Theory*, 2nd ed., chapter 4. Wiley.
- Krejtz, K., Duchowski, A., Szmidt, T., Krejtz, I., Gonzalez Perilli, F., Pires, A., Vilaro, A., & Villalobos, N. (2015). Gaze transition entropy. *ACM Transactions on Applied Perception*, 13(1), 4:1-4:20. doi:[10.1145/2834121](https://doi.org/10.1145/2834121)
- Krejtz, K., Hughes, C.J., Stasiak, I., Duchowski, A., & Krejtz, I. (2025). Real-time mobile transition matrix entropy based on eye and head movements. *Proceedings of ETRA '25*. doi:[10.1145/3715669.3723128](https://doi.org/10.1145/3715669.3723128)
- Shannon, C.E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27, 379-423.

See Also

[entropy_network](#) for the edge-level decomposition, [entropy_trajectory](#) for the sliding-window version, [entropy_bayes](#) for credible intervals; [markov_stability](#), [passage_time](#), [markov_order_test](#), [chain_structure](#)

Examples

```
net <- build_network(as.data.frame(trajectories), method = "relative")
te <- transition_entropy(net)
print(te)
summary(te)
plot(te)
```

validate_netobject	Validate a netobject / cograph_network against the shared schema
--------------------	--

Description

Enforces the structural contract that both `Nestimate` netobjects and **psychnets** objects must satisfy to be interchangeable across the package boundary. This is the single place that says what "a network object" means, so a drift on either side (a renamed field, a mistyped edge column) fails loudly here rather than mis-rendering three layers downstream.

Usage

```
validate_netobject(x)
```

Arguments

x	An object expected to satisfy the cograph_network contract.
---	---

Details

The contract is deliberately the *shared* subset: the `$nodes` x/y layout columns and the `Nestimate` pipeline fields (`$data`, `$level`, ...) are not required, and `$edges` endpoints may be either integer node indices (`Nestimate`) or character labels (`psychnet`).

Value

Invisibly TRUE if `x` conforms; otherwise stops with the full list of violations.

See Also

[as_netobject](#)

Examples

```
net <- build_cor(data.frame(a = rnorm(50), b = rnorm(50), c = rnorm(50)))
validate_netobject(net)
```

verify_simplicial	<i>Verify Simplicial Complex Against igraph</i>
-------------------	---

Description

Cross-validates clique finding and Betti numbers against `igraph` and known topological invariants. Useful for testing.

Usage

```
verify_simplicial(mat, threshold = 0)
```

Arguments

<code>mat</code>	A square adjacency matrix.
<code>threshold</code>	Edge weight threshold.

Value

Invisibly, a list with `cliques_match` (logical: do the simplices match `igraph::cliques()` exactly), `n_simplices_ours`, `n_simplices_igraph`, `betti`, `euler`, and `f_vector`. The comparison is also printed to the console.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
verify_simplicial(mat, threshold = 0.3)
```

vertex_bootstrap

Vertex Bootstrap for Network-Level Statistics

Description

Non-parametric vertex bootstrap of a single observed network (Snijders & Borgatti 1999). Vertices are resampled with replacement and the weight matrix is rebuilt from the original entries of the resampled vertex pairs; network-level statistics computed on each replicate give bootstrap distributions, standard errors, and confidence intervals.

Unlike [bootstrap_network](#), which resamples the underlying cases (sequences or rows) and therefore requires the raw data stored in the netobject, the vertex bootstrap needs **only the weight matrix**. It works on any netobject - including data-less ones such as [build_m1var](#) constituents or `as_tna(mcml)` elements - and on plain weight matrices. The two procedures answer different questions: the case bootstrap quantifies sampling-of-subjects uncertainty in the edge weights; the vertex bootstrap quantifies structural uncertainty of whole-network descriptives given the one network you observed.

Usage

```
vertex_bootstrap(
  x,
  iter = 1000L,
  ci_level = 0.05,
  ci_method = c("percentile", "basic"),
  statistics = NULL,
  statistic_fn = NULL,
  directed = NULL,
  seed = NULL
)

## S3 method for class 'net_vertex_bootstrap'
print(x, digits = 3, ...)

## S3 method for class 'net_vertex_bootstrap'
summary(object, ...)

## S3 method for class 'net_vertex_bootstrap'
plot(x, bins = 30, ...)
```

Arguments

<code>x</code>	A netobject (from build_network or any builder), a <code>cograph_network</code> , or a square numeric weight matrix. For the <code>print()</code> and <code>plot()</code> methods: an object of class <code>net_vertex_bootstrap</code> .
<code>iter</code>	Integer. Number of bootstrap replicates (default 1000).

ci_level	Numeric. Significance level for the confidence intervals (default 0.05 for 95% CIs).
ci_method	Character. "percentile" (default) for empirical quantile intervals, or "basic" for intervals reflected around the observed value (Davison & Hinkley 1997, eq. 5.6).
statistics	Character vector selecting built-in statistics (see Details). Default: all applicable to the network's directedness.
statistic_fn	Optional named list of functions, each taking the weight matrix and returning a single numeric value. Computed alongside the built-ins.
directed	Logical or NULL. Directedness of the network. NULL (default) reads x\$directed when available, otherwise falls back to matrix symmetry.
seed	Integer or NULL. RNG seed for reproducibility.
digits	Number of digits to display (default 3).
...	In plot.net_vertex_bootstrap(), print.net_vertex_bootstrap() and summary.net_vertex_bootstrap(). Additional arguments (ignored).
object	For the summary() method: an object of class net_vertex_bootstrap.
bins	Number of histogram bins (default 30).

Details

Each replicate draws n vertex indices with replacement and sets $W_b[i, j] = W[idx_i, idx_j]$. When the same original vertex is drawn for two different positions, the off-diagonal cell would be a structural self-pair; following Snijders & Borgatti, such cells are filled with the weight of a randomly chosen pair of distinct original vertices. Diagonal entries carry the original self-weight of the resampled vertex ($W[idx_i, idx_i]$) - self-loops are meaningful in transition networks and are never altered. For undirected networks the substitution is applied symmetrically so replicates stay symmetric.

Built-in statistics (all computed on the off-diagonal part of the weight matrix):

density Proportion of non-zero off-diagonal cells.

mean_weight Mean of the non-zero off-diagonal weights.

centralization Freeman-type strength centralization: $\sum(\max(s) - s) / ((n - 1) * \max(s))$ where s is total node strength on absolute weights. 0 when all nodes have equal strength, approaching 1 for a star.

reciprocity Directed networks only. Weighted reciprocity $\sum(\min(|W|, |t(W)|)) / \sum(|W|)$ over off-diagonal cells: the proportion of total weight that is reciprocated.

Value

An object of class "net_vertex_bootstrap" containing:

summary Tidy data frame, one row per statistic: statistic, observed, boot_mean, boot_sd, bias, ci_lower, ci_upper.

boot_stats iter x n_statistics matrix of replicate values.

observed Named vector of observed statistics.

iter, ci_level, ci_method, directed, n_nodes Configuration.

In `print.net_vertex_bootstrap()`: x, invisibly.

In `summary.net_vertex_bootstrap()`: The tidy summary data frame (one row per statistic).

In `plot.net_vertex_bootstrap()`: A ggplot object.

Methods

- `plot.net_vertex_bootstrap()`: Histogram of the bootstrap distribution per statistic, with the observed value (solid line) and confidence bounds (dashed lines).

References

Snijders, T. A. B., & Borgatti, S. P. (1999). Non-parametric standard errors and tests for network statistics. *Connections*, 22(2), 161-170.

Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap Methods and their Application*. Cambridge University Press.

See Also

[bootstrap_network](#) for case-resampling edge-weight inference, [centrality_stability](#) for case-dropping centrality stability.

Examples

```
seqs <- data.frame(
  T1 = c("plan", "code", "debug", "plan", "test", "code"),
  T2 = c("code", "debug", "code", "plan", "code", "test"),
  T3 = c("debug", "code", "plan", "code", "debug", "plan"),
  T4 = c("test", "plan", "test", "debug", "plan", "code")
)
net <- build_network(seqs, method = "relative")
vb <- vertex_bootstrap(net, iter = 100, seed = 1)
summary(vb)

plot(vb)
```

Description

Snijders & Borgatti (1999) two-network test: each network's statistics get vertex-bootstrap standard errors, and each difference is tested with

$$z = (\hat{\theta}_x - \hat{\theta}_y) / \sqrt{SE_x^2 + SE_y^2}$$

against a standard normal reference. This is the comparison the vertex bootstrap was originally proposed for: deciding whether two observed networks differ in density, centralization, reciprocity, or any other whole-network descriptive.

Usage

```
vertex_compare(
  x,
  y,
  iter = 1000L,
  ci_level = 0.05,
  statistics = NULL,
  statistic_fn = NULL,
  directed = NULL,
  seed = NULL,
  labels = c("x", "y")
)

## S3 method for class 'net_vertex_comparison'
print(x, digits = 3, ...)

## S3 method for class 'net_vertex_comparison'
summary(object, ...)

## S3 method for class 'net_vertex_comparison'
plot(x, ...)
```

Arguments

x, y	The two networks: netobjects, cograph_networks, square weight matrices, or precomputed net_vertex_bootstrap objects (then iter, statistics, statistic_fn, directed, and seed are ignored for that argument). For the print() and plot() methods, x is an object of class net_vertex_comparison. The two sides must cover exactly the same statistics; a mismatch (e.g., a directed network's reciprocity against an undirected one, or precomputed objects built with different statistics selections) is an error, never a silent subset.
iter	Integer. Number of bootstrap replicates (default 1000).
ci_level	Numeric. Significance level for the confidence intervals (default 0.05 for 95% CIs).
statistics	Character vector selecting built-in statistics (see Details). Default: all applicable to the network's directedness.

<code>statistic_fn</code>	Optional named list of functions, each taking the weight matrix and returning a single numeric value. Computed alongside the built-ins.
<code>directed</code>	Logical or NULL. Directedness of the network. NULL (default) reads <code>x\$directed</code> when available, otherwise falls back to matrix symmetry.
<code>seed</code>	Integer or NULL. RNG seed for reproducibility.
<code>labels</code>	Character vector of length 2 naming the networks in the output (default <code>c("x", "y")</code>).
<code>digits</code>	Number of digits to display (default 3).
<code>...</code>	In <code>plot.net_vertex_comparison()</code> , <code>print.net_vertex_comparison()</code> and <code>summary.net_vertex_comparison()</code> : Additional arguments (ignored).
<code>object</code>	For the <code>summary()</code> method: an object of class <code>net_vertex_comparison</code> .

Value

An object of class "net_vertex_comparison" containing:

summary Tidy data frame, one row per statistic: `statistic`, the two observed values, `diff`, `se_diff`, `z`, `p_value`, and a normal-approximation confidence interval for the difference.

x, y The two `net_vertex_bootstrap` results.

labels, ci_level Configuration.

When both bootstrap SEs are zero (a statistic with no resampling variation in either network) `z` and `p_value` are NA.

In `print.net_vertex_comparison()`: `x`, invisibly.

In `summary.net_vertex_comparison()`: The tidy summary data frame (one row per statistic).

In `plot.net_vertex_comparison()`: A ggplot object.

Methods

- `plot.net_vertex_comparison()`: Forest plot of the statistic differences with normal-approximation confidence intervals; differences whose interval excludes zero are the statistically distinguishable ones.

References

Snijders, T. A. B., & Borgatti, S. P. (1999). Non-parametric standard errors and tests for network statistics. *Connections*, 22(2), 161-170.

See Also

[vertex_bootstrap](#), [nct](#) for the permutation-based comparison of edge-level structure when raw data are available, [permutation](#).

Examples

```

states <- c("plan", "code", "debug", "test")
s1 <- data.frame(
  T1 = rep(states, 5), T2 = rep(rev(states), 5),
  T3 = rep(states[c(2, 3, 4, 1)], 5)
)
s2 <- data.frame(
  T1 = rep(states[c(3, 1, 4, 2)], 5), T2 = rep(states, 5),
  T3 = rep(states[c(4, 3, 1, 2)], 5)
)
net1 <- build_network(s1, method = "relative")
net2 <- build_network(s2, method = "relative")
cmp <- vertex_compare(net1, net2, iter = 100, seed = 1)
summary(cmp)

```

wide_to_long

*Convert Wide Sequences to Long Format***Description**

Convert sequence data from wide format (one row per sequence, columns as time points) to long format (one row per action).

Usage

```

wide_to_long(
  data,
  id_col = NULL,
  time_prefix = "V",
  action_col = "Action",
  time_col = "Time",
  drop_na = TRUE
)

```

Arguments

<code>data</code>	Data frame in wide format with sequences in rows.
<code>id_col</code>	Character. Name of the ID column, or NULL to auto-generate IDs. Default: NULL.
<code>time_prefix</code>	Character. Prefix for time point columns (e.g., "V" for V1, V2, ...). Default: "V".
<code>action_col</code>	Character. Name of the action column in output. Default: "Action".
<code>time_col</code>	Character. Name of the time column in output. Default: "Time".
<code>drop_na</code>	Logical. Whether to drop NA values. Default: TRUE.

Details

Converts wide sequence data (one row per sequence, one column per time point) to the long format used by many TNA functions and analyses.

Value

A data frame in long format, one row per (sequence, time point), sorted by identifier then time, with columns:

id Sequence identifier. Named by `id_col`; when that is `NULL` the column is called `id` and holds the row number of the wide input (integer).

Time Time point within the sequence (integer), taken from the numeric suffix of the wide column name. Named by `time_col`.

Action The action/state at that time point. Named by `action_col`.

Any additional non-time columns from the original data are preserved and repeated on every row of their sequence.

See Also

[long_to_wide](#) for the reverse conversion, [prepare_for_tna](#) for preparing data for TNA analysis.

Examples

```
wide_data <- data.frame(
  V1 = c("A", "B", "C"), V2 = c("B", "C", "A"), V3 = c("C", "A", "B")
)
long_data <- wide_to_long(wide_data)
head(long_data)
```

 wtna

Window-based Transition Network Analysis

Description

Computes networks from one-hot (binary indicator) data using temporal windowing. Supports transition (directed), co-occurrence (undirected), or both network types.

Usage

```
wtna(
  data,
  method = c("transition", "cooccurrence", "both"),
  type = c("frequency", "relative"),
  codes = NULL,
  window_size = 3L,
  mode = c("non-overlapping", "overlapping"),
```

```

    actor = NULL
  )

## S3 method for class 'wtna_mixed'
print(x, ...)
```

Arguments

<code>data</code>	Data frame with one-hot encoded columns (0/1 binary).
<code>method</code>	Character. Network type: "transition" (directed), "cooccurrence" (undirected), or "both" (returns list of two networks). Default: "transition".
<code>type</code>	Character. Output type: "frequency" (raw counts) or "relative" (row-normalized probabilities). Default: "frequency". Note that type = "relative" applied to method = "cooccurrence" produces an asymmetric matrix (conditional co-occurrence given row state), not a symmetric undirected weight matrix - use type = "frequency" if symmetric co-occurrence counts are required.
<code>codes</code>	Character vector or NULL. Names of the one-hot columns to use. If NULL, auto-detects binary columns. Default: NULL.
<code>window_size</code>	Integer (≥ 1). Number of consecutive rows to aggregate per window. Default: 3 (windowed pairwise between-window counting). Set window_size = 1 for ordinary consecutive ($t \rightarrow t+1$) transitions with no windowing.
<code>mode</code>	Character. Window mode: "non-overlapping" (fixed, separate windows) or "overlapping" (rolling, step = 1). Default: "non-overlapping".
<code>actor</code>	Character or NULL. Name of the actor/ID column for per-group computation. If NULL, treats all rows as one group. Default: NULL.
<code>x</code>	For the print() method: an object of class wtna_mixed.
<code>...</code>	In print.wtna_mixed(): Additional arguments (ignored).

Details

Transitions: Uses `crossprod(X[-n, ], X[-1, ])` to count how often state i is active at time t AND state j at time $t+1$.

Co-occurrence: Uses `crossprod(X)` to count states that are simultaneously active in the same row.

Windowing: For `window_size > 1`, rows are aggregated into windows before computing networks. Non-overlapping windows are fixed, separate blocks; overlapping windows roll forward one row at a time. Within each window, any active indicator (1) in any row makes that state active for the window.

Per-actor: When `actor` is specified, networks are computed per group and summed.

Value

For `method = "transition"` or `"cooccurrence"`: a `c("netobject", "cograph_network")` object (see [build_network](#)) with `method` set to `"wtna_transition"` or `"wtna_cooccurrence"`, `directed = TRUE` only for transitions, and the windowing settings (`type`, `window_size`, `mode`, `codes`, `actor`) recorded in `$params`. Transition networks also carry `$initial`, the per-actor-averaged initial state distribution.

For `method = "both"`: a `wtna_mixed` object - a list with elements `$transition` and `$cooccurrence` (each a `netobject` as above) and `$method = "wtna_both"`.

In `print.wtna_mixed()`: The input object, invisibly.

See Also

[build_network](#), [prepare_onehot](#)

Examples

```
oh <- matrix(c(1,0,0, 0,1,0, 0,0,1, 1,0,0), nrow = 4, byrow = TRUE,
             dimnames = list(NULL, c("A","B","C")))
w <- wtna(oh)

# Simple one-hot data
df <- data.frame(
  A = c(1, 0, 1, 0, 1),
  B = c(0, 1, 0, 1, 0),
  C = c(0, 0, 1, 0, 0)
)

# Transition network
net <- wtna(df)
print(net)

# Both networks
nets <- wtna(df, method = "both")
print(nets$transition)
print(nets$cooccurrence)

# With windowing
net <- wtna(df, window_size = 2, mode = "non-overlapping")
```

Index

* datasets

- chatgpt_srl, 94
 - group_regulation_long, 146
 - learning_activities, 158
 - long-data, 161
 - srl_strategies, 248
 - trajectories, 253
-
- action_to_onehot, 5, 231
 - actor_endpoints, 6, 253
 - actor_endpoints(), 171, 172
 - ai_long (long-data), 161
 - as.data.frame(), 155
 - as.data.frame.net_cluster_diagnostics (cluster_diagnostics), 99
 - as.data.frame.net_hypergraph_cluster (hypergraph_cluster), 149
 - as.data.frame.net_hypergraph_transduction (hypergraph_transduction), 154
 - as.data.frame.state_freq (plot_state_frequencies), 217
 - as_htna, 9, 55
 - as_netdifference, 11, 19
 - as_netobject, 12, 257
 - as_networks, 13, 16
 - as_tna, 13, 14, 14, 55, 107, 164
 - association_rules, 7
-
- bayes_compare, 16, 90, 131, 210
 - bayes_compare(), 117, 118
 - betti_numbers, 19, 80, 248
 - bipartite_groups, 20
 - bipartite_groups(), 95, 96, 148, 149, 151–154
 - boot_glasso, 25
 - bootstrap_network, 10, 21, 28, 60, 61, 76, 88–90, 124, 182, 183, 192, 210, 258, 260
 - bootstrap_network(), 63, 64, 84
 - bottleneck_distance, 29
-
- build_atna, 30
 - build_clusters, 31, 66, 68, 97, 98, 102–104, 128, 239, 240, 243–245
 - build_cna, 35, 125
 - build_cor, 36
 - build_ftna, 37
 - build_gimme, 38
 - build_glasso, 40
 - build_hon, 41, 44
 - build_honem, 44
 - build_hypa, 46
 - build_hypergraph, 49
 - build_hypergraph(), 20, 21, 95, 96, 148, 149, 152–154
 - build_ising, 51
 - build_mcml, 9, 11, 15, 16, 52, 56, 61, 105, 107, 164, 240, 243
 - build_mcml_pc, 13, 14, 56, 120, 121, 157, 159, 163, 164
 - build_mlvar, 62, 258
 - build_mlvar(), 108
 - build_mmm, 65, 102, 103, 110, 240, 244, 245
 - build_mogen, 69, 204
 - build_network, 9–11, 16, 17, 19, 21, 22, 25, 28, 30, 32, 35–37, 40, 41, 51, 58, 68, 71, 77, 78, 81, 86, 88, 89, 102–105, 124, 133, 136, 137, 140, 162–164, 183, 199, 205, 206, 210, 226, 228, 230, 233, 239, 244, 245, 258, 265, 266
 - build_network(), 51, 64, 93, 96, 165, 172
 - build_pcor, 77
 - build_simplicial, 78, 248
 - build_simplicial(), 50, 51
 - build_tna, 80
-
- casedrop_reliability, 81
 - centrality_stability, 85, 260
 - centrality_stability(), 64, 83, 84
 - certainty, 19, 25, 88

- certainty(), [64](#)
- chain_structure, [90](#), [254](#), [256](#)
- chain_structure(), [6](#), [172](#)
- chatgpt_srl, [94](#)
- clique_expansion, [95](#)
- clique_expansion(), [148](#), [149](#), [154](#)
- cluster_choice, [96](#)
- cluster_diagnostics, [98](#), [99](#)
- cluster_mmm, [68](#), [76](#), [101](#), [104](#)
- cluster_network, [68](#), [76](#), [102](#), [103](#), [239](#)
- cluster_summary, [15](#), [16](#), [52](#), [53](#), [55](#), [104](#)
- coefs, [108](#)
- coefs(), [63](#), [64](#)
- compare_mmm, [68](#), [96](#), [98](#), [100](#), [109](#)
- compare_model, [111](#)
- compare_model(), [118](#), [165](#)
- compare_networks, [114](#), [114](#)
- compare_networks(), [119](#)
- comparison_tables, [119](#)
- composites, [120](#), [157](#)
- convert_sequence_format, [21](#), [59](#), [121](#), [145](#)
- cooccurrence, [35](#), [123](#)
- cooccurrence(), [95](#)

- distribution_plot, [125](#), [238](#), [240](#), [242](#), [243](#), [245](#)

- edge_differences (comparison_tables), [119](#)
- edge_differences(), [117](#)
- effects_table, [128](#), [196](#)
- entropy_bayes, [129](#), [133](#), [136](#), [256](#)
- entropy_network, [129](#), [131](#), [131](#), [256](#)
- entropy_trajectory, [134](#), [256](#)
- estimate_network, [136](#)
- euler_characteristic, [138](#)
- evaluate_links, [138](#), [226](#)
- extract_edges, [139](#), [141](#), [144](#)
- extract_initial_probs, [140](#), [144](#)
- extract_pathways, [141](#)
- extract_transition_matrix, [140](#), [141](#), [143](#)

- frequencies, [122](#), [144](#)

- get_estimator, [146](#), [159](#), [233](#)
- glm, [195](#)
- global_differences (comparison_tables), [119](#)
- global_differences(), [117](#)

- group_regulation_long, [146](#)

- human_long (long-data), [161](#)
- hypergraph_centrality, [147](#)
- hypergraph_cluster, [149](#)
- hypergraph_laplacian, [151](#)
- hypergraph_laplacian(), [150](#), [155](#)
- hypergraph_measures, [152](#)
- hypergraph_measures(), [149](#)
- hypergraph_transduction, [154](#)

- idiographic::fit_gimme(), [38–40](#)
- idiographic::fit_mlvar(), [63](#)
- item_loadings, [60](#), [121](#), [157](#)

- learning_activities, [147](#), [158](#)
- list_estimators, [76](#), [146](#), [159](#), [233](#), [234](#)
- loading_stability, [60](#), [61](#), [157](#), [159](#)
- long-data, [161](#)
- long_to_wide, [162](#), [229](#), [264](#)

- macro_network, [15](#), [16](#), [55](#), [107](#), [163](#)
- magnitude_difference, [164](#)
- mark_first_state, [171](#)
- mark_terminal_state, [30](#), [35](#), [37](#), [74](#), [81](#), [172](#), [253](#)
- mark_terminal_state(), [6](#), [171](#)
- markov_order_test, [166](#), [204](#), [256](#)
- markov_stability, [169](#), [199](#), [256](#)
- markov_stability(), [91](#), [93](#)
- mogen_transitions, [70](#), [173](#)
- mosaic_analysis, [174](#), [179](#)
- mosaic_plot, [176](#), [176](#), [220](#)

- nct, [180](#), [262](#)
- net_aggregate_weights, [184](#)
- net_centrality, [13](#), [185](#)
- net_deprune, [188](#), [192](#), [194](#)
- net_edge_betweenness, [16](#), [17](#), [189](#), [190](#), [206](#), [209](#), [252](#)
- net_hypergraph, [20](#), [95](#), [147](#), [152](#)
- net_prune, [188](#), [190](#), [193](#), [194](#)
- net_pruning_details, [192](#), [193](#)
- net_reprune, [188](#), [192](#), [194](#)
- netobject, [95](#)
- network_metrics (comparison_tables), [119](#)
- network_metrics(), [117](#)
- network_reliability, [88](#), [90](#), [182](#)
- network_reliability(), [64](#)

- node_differences (comparison_tables), 119
- node_differences(), 117
- OlsonNames, 227
- outcome_model, 128, 142, 195, 248
- p.adjust, 196, 206, 236, 237
- p.adjust.methods, 47
- passage_time, 170, 197, 256
- passage_time(), 91, 93
- path_counts, 201
- path_dependence, 202
- pathways, 199
- permutation, 10, 16, 18, 19, 24, 124, 190, 205, 210–212, 262
- permutation(), 116–118
- permutation_diagnostics, 207, 210, 210
- persistence_landscape, 213
- persistent_homology, 80, 214
- plot, 45, 70
- plot.boot_glasso (boot_glasso), 25
- plot.chain_structure (chain_structure), 90
- plot.cluster_choice (cluster_choice), 96
- plot.magnitude_difference (magnitude_difference), 164
- plot.mcml_pc (build_mcml_pc), 56
- plot.mmm_compare (compare_mmm), 109
- plot.mosaic_analysis (mosaic_analysis), 174
- plot.net_association_rules (association_rules), 7
- plot.net_bayes (bayes_compare), 16
- plot.net_casedrop_reliability (casedrop_reliability), 81
- plot.net_casedrop_reliability_group (casedrop_reliability), 81
- plot.net_centrality (net_centrality), 185
- plot.net_centrality_group (net_centrality), 185
- plot.net_cluster_diagnostics, 100
- plot.net_cluster_diagnostics (cluster_diagnostics), 99
- plot.net_clustering, 68, 100
- plot.net_clustering (build_clusters), 31
- plot.net_comparison (compare_model), 111
- plot.net_edge_betweenness (net_edge_betweenness), 189
- plot.net_entropy_bayes (entropy_bayes), 129
- plot.net_entropy_trajectory (entropy_trajectory), 134
- plot.net_honem (build_honem), 44
- plot.net_hypergraph_cluster (hypergraph_cluster), 149
- plot.net_hypergraph_transduction (hypergraph_transduction), 154
- plot.net_markov_order (markov_order_test), 166
- plot.net_markov_stability (markov_stability), 169
- plot.net_mmm, 100
- plot.net_mmm (build_mmm), 65
- plot.net_mmm_clustering, 100
- plot.net_mmm_clustering (build_mmm), 65
- plot.net_mogen (build_mogen), 69
- plot.net_mpt (passage_time), 197
- plot.net_network_comparison (compare_networks), 114
- plot.net_outcome_model (outcome_model), 195
- plot.net_path_dependence (path_dependence), 202
- plot.net_reliability (network_reliability), 182
- plot.net_sequence_comparison (sequence_compare), 235
- plot.net_stability (centrality_stability), 85
- plot.net_transition_entropy (transition_entropy), 253
- plot.net_vertex_bootstrap (vertex_bootstrap), 258
- plot.net_vertex_comparison (vertex_compare), 260
- plot.pc_loading_stability (loading_stability), 159
- plot.persistence_landscape (persistence_landscape), 213
- plot.persistent_homology (persistent_homology), 214
- plot.q_analysis (q_analysis), 231
- plot.simplicial_complex (build_simplicial), 78

plot.state_freq
 (plot_state_frequencies), 217
 plot_mosaic, 180, 216
 plot_state_frequencies, 217, 217, 245, 251
 predict_links, 9, 224
 predictability, 222
 prepare, 74, 167, 226
 prepare_for_tna, 163, 228, 264
 prepare_onehot, 228, 230, 266
 print.boot_glasso (boot_glasso), 25
 print.chain_structure
 (chain_structure), 90
 print.chain_structure_group
 (chain_structure), 90
 print.cluster_choice (cluster_choice), 96
 print.hypergraph_measures
 (hypergraph_measures), 152
 print.magnitude_difference
 (magnitude_difference), 164
 print.mcml (build_mcml), 52
 print.mcml_layer (build_mcml), 52
 print.mcml_pc (build_mcml_pc), 56
 print.mcml_sequence_plot
 (sequence_plot), 238
 print.mmm_compare (compare_mmm), 109
 print.mosaic_analysis
 (mosaic_analysis), 174
 print.nestimate_data (prepare), 226
 print.nestimate_facet_list
 (plot_state_frequencies), 217
 print.nestimate_facet_plot
 (plot_state_frequencies), 217
 print.net_association_rules
 (association_rules), 7
 print.net_bayes (bayes_compare), 16
 print.net_bayes_group (bayes_compare), 16
 print.net_bootstrap, 25
 print.net_bootstrap
 (bootstrap_network), 21
 print.net_bootstrap_group
 (bootstrap_network), 21
 print.net_casedrop_reliability
 (casedrop_reliability), 81
 print.net_casedrop_reliability_group
 (casedrop_reliability), 81
 print.net_certainty (certainty), 88
 print.net_cluster_diagnostics, 100
 print.net_cluster_diagnostics
 (cluster_diagnostics), 99
 print.net_clustering, 68, 76, 100
 print.net_clustering (build_clusters), 31
 print.net_comparison (compare_model), 111
 print.net_entropy_bayes
 (entropy_bayes), 129
 print.net_entropy_bayes_group
 (entropy_bayes), 129
 print.net_entropy_trajectory
 (entropy_trajectory), 134
 print.net_hon (build_hon), 41
 print.net_honem (build_honem), 44
 print.net_hypa (build_hypa), 46
 print.net_hypergraph
 (build_hypergraph), 49
 print.net_hypergraph_cluster
 (hypergraph_cluster), 149
 print.net_hypergraph_transduction
 (hypergraph_transduction), 154
 print.net_link_prediction
 (predict_links), 224
 print.net_markov_order
 (markov_order_test), 166
 print.net_markov_order_group
 (markov_order_test), 166
 print.net_markov_stability
 (markov_stability), 169
 print.net_markov_stability_group
 (markov_stability), 169
 print.net_mlvar (build_mlvar), 62
 print.net_mmm, 76, 100
 print.net_mmm (build_mmm), 65
 print.net_mmm_clustering (build_mmm), 65
 print.net_mogen (build_mogen), 69
 print.net_mpt (passage_time), 197
 print.net_mpt_group (passage_time), 197
 print.net_nct (nct), 180
 print.net_network_comparison
 (compare_networks), 114
 print.net_outcome_model
 (outcome_model), 195
 print.net_path_dependence
 (path_dependence), 202

- `print.net_permutation`, 210
- `print.net_permutation(permutation)`, 205
- `print.net_permutation_group(permutation)`, 205
- `print.net_pruning_details(net_pruning_details)`, 193
- `print.net_reliability(network_reliability)`, 182
- `print.net_sequence_comparison(sequence_compare)`, 235
- `print.net_stability(centrality_stability)`, 85
- `print.net_stability_group(centrality_stability)`, 85
- `print.net_table(comparison_tables)`, 119
- `print.net_transition_entropy(transition_entropy)`, 253
- `print.net_transition_entropy_group(transition_entropy)`, 253
- `print.net_vertex_bootstrap(vertex_bootstrap)`, 258
- `print.net_vertex_comparison(vertex_compare)`, 260
- `print.netdifference(subtract_networks)`, 252
- `print.netobject(build_network)`, 71
- `print.netobject_group(build_network)`, 71
- `print.netobject_ml(build_network)`, 71
- `print.pc_loading_stability(loading_stability)`, 159
- `print.persistence_landscape(persistence_landscape)`, 213
- `print.persistent_homology(persistent_homology)`, 214
- `print.q_analysis(q_analysis)`, 231
- `print.simplicial_complex(build_simplicial)`, 78
- `print.state_freq(plot_state_frequencies)`, 217
- `print.summary.net_casedrop_reliability_group(casedrop_reliability)`, 81
- `print.summary.net_mpt(passage_time)`, 197
- `print.summary.net_path_dependence(path_dependence)`, 202
- `print.summary.net_transition_entropy(transition_entropy)`, 253
- `print.summary.netobject(build_network)`, 71
- `print.summary.netobject_group(build_network)`, 71
- `print.summary_chain_structure(chain_structure)`, 90
- `print.tidy_covariates(build_clusters)`, 31
- `print.wtna_boot_mixed(bootstrap_network)`, 21
- `print.wtna_mixed(wtna)`, 264
- `print.wtna_perm_mixed(permutation)`, 205
- `q_analysis`, 80, 231, 248
- `register_estimator`, 76, 146, 159, 233, 234
- `reliability`, 89
- `remove_estimator`, 233, 234
- `rename_models`, 234
- `sequence_compare`, 142, 235
- `sequence_plot`, 54, 55, 125–128, 238, 243, 245
- `session_ids`, 34, 67, 244
- `set_state_colors`, 245, 249, 250
- `simplicial_degree`, 80, 247
- `simplicial_features`, 195, 196, 247
- `srl_strategies`, 147, 248
- `state_colors`, 246, 249
- `state_colors<- (set_state_colors)`, 245
- `state_distribution`, 250
- `state_freq(plot_state_frequencies)`, 217
- `state_frequencies`, 250, 251
- `stats::p.adjust()`, 116
- `subtract_networks`, 19, 252
- `subtract_networks()`, 118
- `summary.boot_glasso(boot_glasso)`, 25
- `summary.chain_structure(chain_structure)`, 90
- `summary.chain_structure_group(chain_structure)`, 90
- `summary.cluster_choice(cluster_choice)`, 96
- `summary.mcml(build_mcml)`, 52
- `summary.mcml_pc(build_mcml_pc)`, 56
- `summary.mmm_compare(compare_mmm)`, 109
- `summary.mosaic_analysis(mosaic_analysis)`, 174

- summary.nest_initial_probs
(extract_initial_probs), 140
- summary.nest_transition_counts
(frequencies), 144
- summary.nest_transition_matrix
(extract_transition_matrix), 143
- summary.net_association_rules
(association_rules), 7
- summary.net_bayes (bayes_compare), 16
- summary.net_bayes_group
(bayes_compare), 16
- summary.net_bootstrap, 24, 25
- summary.net_bootstrap
(bootstrap_network), 21
- summary.net_bootstrap_group
(bootstrap_network), 21
- summary.net_casedrop_reliability
(casedrop_reliability), 81
- summary.net_casedrop_reliability_group
(casedrop_reliability), 81
- summary.net_clustering
(build_clusters), 31
- summary.net_entropy_bayes
(entropy_bayes), 129
- summary.net_entropy_trajectory
(entropy_trajectory), 134
- summary.net_hon (build_hon), 41
- summary.net_honem (build_honem), 44
- summary.net_hypa (build_hypa), 46
- summary.net_hypergraph
(build_hypergraph), 49
- summary.net_hypergraph_cluster
(hypergraph_cluster), 149
- summary.net_hypergraph_transduction
(hypergraph_transduction), 154
- summary.net_link_prediction
(predict_links), 224
- summary.net_markov_order
(markov_order_test), 166
- summary.net_markov_stability
(markov_stability), 169
- summary.net_mlvar (build_mlvar), 62
- summary.net_mmm (build_mmm), 65
- summary.net_mogen (build_mogen), 69
- summary.net_mpt (passage_time), 197
- summary.net_nct (nct), 180
- summary.net_network_comparison
(comparison_tables), 119
- summary.net_outcome_model
(outcome_model), 195
- summary.net_path_dependence
(path_dependence), 202
- summary.net_permutation, 208, 210
- summary.net_permutation (permutation), 205
- summary.net_permutation_group
(permutation), 205
- summary.net_reliability
(network_reliability), 182
- summary.net_sequence_comparison
(sequence_compare), 235
- summary.net_stability
(centrality_stability), 85
- summary.net_stability_group
(centrality_stability), 85
- summary.net_transition_entropy
(transition_entropy), 253
- summary.net_vertex_bootstrap
(vertex_bootstrap), 258
- summary.net_vertex_comparison
(vertex_compare), 260
- summary.netobject (build_network), 71
- summary.netobject_group
(build_network), 71
- summary.wtna_boot_mixed
(bootstrap_network), 21
- summary.wtna_perm_mixed (permutation), 205
- trajectories, 253
- transition_entropy, 129, 131–133, 136, 204, 253
- validate_netobject, 13, 256
- verify_simplicial, 257
- vertex_bootstrap, 60, 61, 258, 262
- vertex_compare, 60, 260
- vertex_compare(), 117, 118
- wide_to_long, 163, 229, 263
- wtna, 264